2013

Mobile Studienet

BPRI1 Project report

AUTHORS
Lubomir Sotirov (142323)

Dimitar Spasov (143062)

SUPERVISOR
Stephan Erbs Korsholm

December 13, 2013

Contents

Abstract 4
Introduction 4
Analysis 6
General analytics 6
Survey results 8
Technical analytics 11
Requirements and use case description 14
Platform availability 22
User experience 22
Delimitations 24
Tools and technologies 24
The backend 24
Development tools 25
The frontend 27
Design and architecture 28
System architecture 28
The backend 29
Database design 29
WCF service and Entity Framework 36
Admin Panel 41
The frontend 43
Android application architecture 43
Implementation 48
The backend 48
Creating an event 48
Fail proofing 54
The frontend 55
Polling data 55
Starting the PollingService 57
Fetching and comparing data from the backend 58
Testing and results 60
General testing 60
White box testing 60

Black box testing 62
Discussion 63
Conclusion 64
Appendices 64
Appendix 1 – Document references 64
Appendix 2 – Detailed use case diagram description 66
Appendix 3 – User manual for the Mobile Studienet system 68

Figure 1 – Web version of Studienet viewed from a laptop at full screen 5
Figure 2 - Web version of Studienet viewed through a mobile device at full screen 5
Figure 3 - Survey results for question number one 8
Figure 4 - Survey results for question number two 9
Figure 5 - Survey results for question number three 9
Figure 6 - Survey results for question number four 10
Figure 7 - Program announcements section 12
Figure 8 - Email section 12
Figure 9 - Courses section 13
Figure 10 - Course announcements section 13
Figure 11 - Session plan section 13
Figure 12 - Session material section 14
Figure 13 - Mobile Studienet use case diagram 15
Figure 14 - The Event sack screen of the application 16
Figure 15 - Detailed event information screen 16
Figure 16 - Android email client synchronized with VIA mail 17
Figure 17 - Notifications of the Mobile Studienet application 17
Figure 18 - Program announcements section of the Mobile Studienet 18
Figure 19 - Calendar screen 18
Figure 20 - Detailed daily schedule 19
Figure 21 – The screen showing all the courses the student participates in 19
Figure 22 - Session plan screen 20
Figure 23 - Course announcements screen 20
Figure 24 - Course info screen 20
Figure 25 - Screen enlisting all teachers and students participating in the course 20
Figure 26 - State chart map of the Mobile Studienet system 21
Figure 27 - Abstract ER model diagram of the database 21
Figure 28 - User interface comparison illustration between the Google Play Store and the Mobile studienet applications 23
Figure 29 - URL link of the web version of Studienet 25
Figure 30 - Size comparison between JSON message and XML message 27
Figure 31 - Mobile Studienet system architecture 28
Figure 32 - Mobile Studienet system workflow 29
Figure 33 - Entity Relationship model diagram 30
Figure 34 - Model View Controller workflow 38

Figure 35 - WCF service architecture 38
Figure 36 - Entity Framework model classes 39
Figure 37 - Entity Framework model diagram 40
Figure 38 - An abstract class diagram of the WCF service project 41
Figure 39 - Admin panel - event manager 41
Figure 40 - This is the Add Event window of the Admin Panel program 42
Figure 41 – Android application abstract class diagram 43
Figure 42 - Google Cloud Messaging workflow 44
Figure 43 - Polling mechanism workflow 45
Figure 44 - Log-in screen 46
Figure 45 - Mobile Studienet main menu 47
Figure 46 - Course menu 48
Figure 47 - Code for creating an event entry in the database 49
Figure 48 - CreateEent function 49
Figure 49 - EventTable model class 50
Figure 50 - Algorithms for the "Applies to" option of the Create event wizard 51
Figure 51 - Assign event to student function 52
Figure 52 - EventStudentBT holding associations between students and events 52
Figure 53 - GetAllEventsForStudent function that is exposed to the network 53
Figure 54 - Full GetAllEventsForStudent function 54
Figure 55 - Message boxes for different warning scenarios 54
Figure 56 - Empty fields warning 55
Figure 57 - Student number character error 55
Figure 58 - Student number length error 55
Figure 59 - PollingService activity diagram 56
Figure 60 - MainMenuActivity onCreate() method 57
Figure 61 - PollingService onStartCommand() method 58
Figure 62 - White box testing 60
Figure 63 - Error report from a program crash 61
Figure 64 - Android SDK LogCat tool 62
Figure 65 – Black box testing 62
Figure 66 - Use case diagram 66
Figure 67 - Log in screen 68
Figure 68 - Main Menu screen 69
Figure 69 - Detailed event screen 69
Figure 70 - Calendar screen 69
Figure 71 - Daily schedule screen 69
Figure 72 – Email redirecting screen 69
Figure 73 - Default android email application 69
Figure 74 - Main menu courses tab 69
Figure 75 - Courses menu 69

Abstract

This documentation includes detailed information concerning all of the stages of development of the Mobile Studienet system, and its design. The Mobile Studienet system's role, is to get the important information from the current web based solution and present it to the end user in a manner suitable for mobile devices. It is aimed to provide an "on the go" way for the students of VIA University College to access their educational information, and improve their overall awareness of the subjects. Any user of this system should be able to access his or hers announcements for the program and the courses he or she is enlisted to. Users will also be able to see the most up to date information for specific topics, receive important notifications and manage their university e-mail account through their mobile device.

Introduction

Telephones, paper letters and telegrams were once world's strongest and most powerful way of communication. Since then humanity has come a very long way, and yesterday's telephones, paper letters and telegrams are today's smartphones, e-mails, and content sharing applications. For a person taking on any type of education, it is of the utmost importance that they are able to access lots of information, in the smallest window of time. VIA University College, is the place where every year thousands of students finish their education and are ready to meet their opportunities in the world. That wouldn't be possible if a thing like the Studienet did not exist. It is the number one place for a student attending the VIA University College. It is the hub that contains all of the materials, presentations, examples and announcements that are essential to the student for his or her education. Up until now, it has only been available, as a web based solution. With today's fast paced evolution of technology and the growing mobile user demand, it comes to mind that we have to step it up a notch.

The problem with today's web based solution (showing on figure 1) of the Studienet is that it is optimized to be accessed only from a computer. But what happens when someone wants to access it while they are running some errands and are outside in the city somewhere. Looking at your small smartphone screen that is trying to fit a full-sized webpage is not just frustrating to look at, but also irritating to use. Missing to read an important e-mail in time, or to notice an important announcement concerning a hand-in due date for example, poses a threat to their study performance, and that is another reason why we need something to keep the students informed, in times like these. A system that could hold the information that Studienet does, and at the same time be so light that you could use it on your smartphone in any time at any place, could harness the potential to keep the students informed at all times, and improve their overall performance in the studies.

This report is structured as follows: first we are presenting a detailed analysis on the research and subjects at hand. Next we are describing the tools used, and the design of the implemented solution. And to finish off we are discussing how we actually implemented the solution and the choices we made for that to happen.

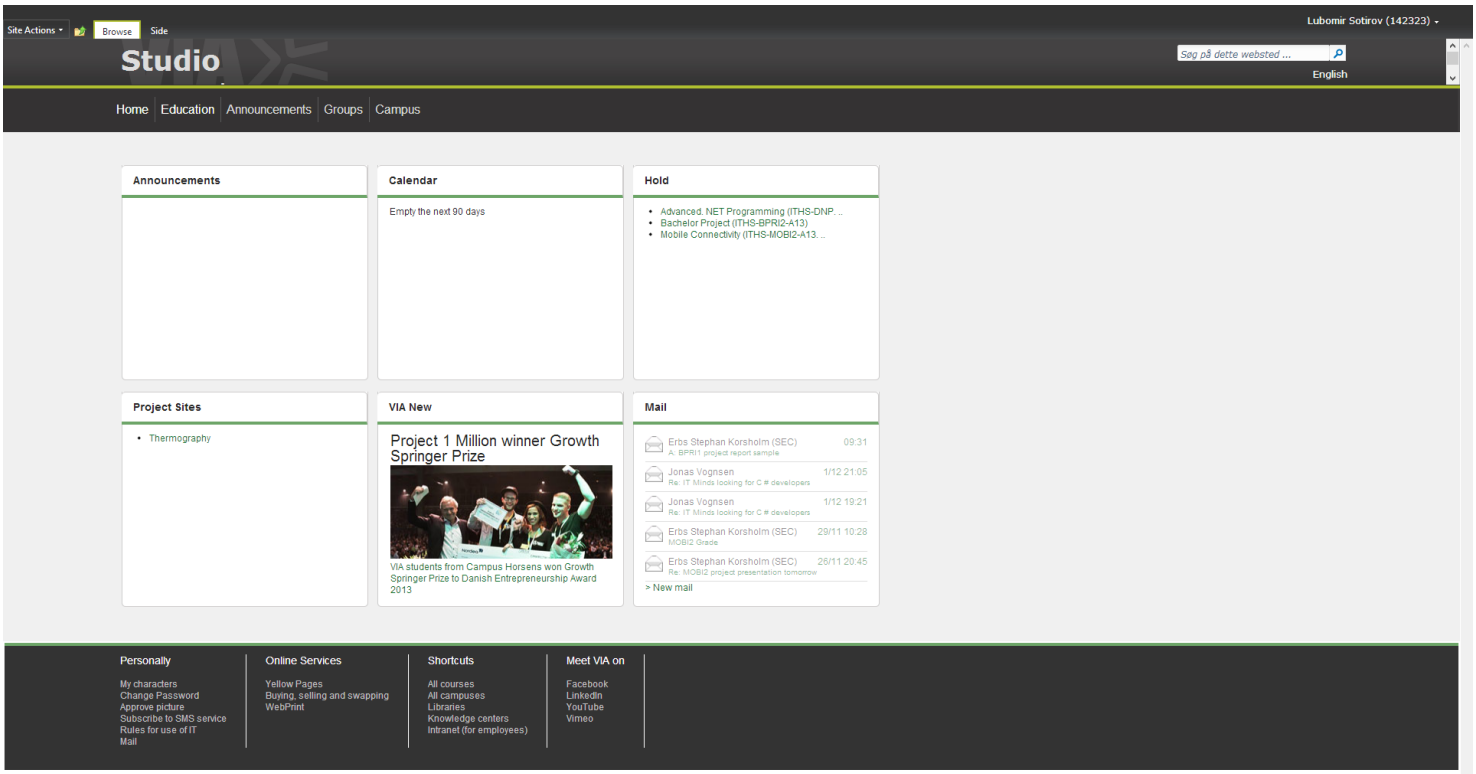


Figure 1 - Web version of Studienet viewed from a laptop at full screen

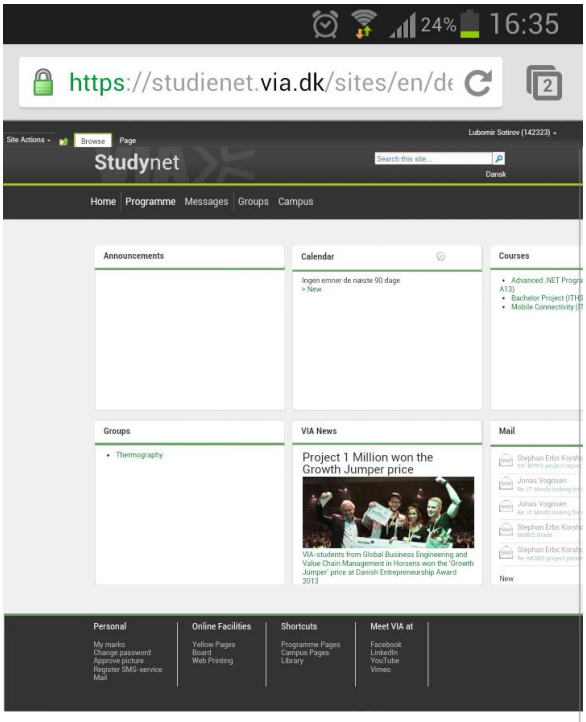


Figure 2 - Web version of Studienet viewed through a mobile device at full screen

Analysis

General analytics

Having in mind all of the problems stated earlier, our team had to focus on them, and think about what would be the best way to approach them, and what would be the rational and most appealing way to simplify and solve them. So to start, we sat down and came up with a list of scenarios which would be most beneficial to the students using the system.

- One thing that could be a benefit when the user is not being able to access a computer at the moment, but they would still like to access the Studienet and see some information. Therefore a fitting mobile solution for the purpose could come in handy.

- Sometimes even if rarely, it happens so that even by looking at the Studienet, students are prone to misreading important announcement or event information. Notifications could be the solution, since the notifications of a mobile device produce a sound and visual signal that the user will most certainly recognize within minutes if not even right away.

- E-mail management on the go is another big plus to the users. Let's say a student receives an e-mail from a teacher concerning a project report that needs to be improved in order to fulfill the basic requirements, but the student is busy with some other task that needs his attention and forgets to check his e-mail. That could as well mean that he can fail at an exam. Being notified on the spot can save a lot of headaches, don't you think?

By analyzing the ordinary habits of an average university student and how they go about using the web version of Studienet, this shed some insight of what the mobile application should consist of and look like. The next step of our analysis of the matter was to construct a rough list of some of the main features we think would be essential to the product.

List of features based on the initial pre-development analysis and scenarios:

- The user should be able to log in to his or hers VIA University College account, in the same way and with the same credentials they use to log into the web version of Studienet – This would allow for easy and seamless transition from the web version to the mobile one, without making it over complicated.

- The user will be able to see his or hers day by day schedule, based on the courses they are enlisted for – That way students will be able to keep track of their schedule and be informed if there are any sudden changes to the classes, if a teacher is sick or the classes are canceled.

- The user will be able to manage his or hers VIA e-mail account through their mobile device (send/receive e-mails) – Having their inbox synchronized with the smartphone would be more comfortable and-user friendly option for the students, than going online and using the online email solution.

- The user will be able to see all the courses he or she is enlisted for and also the session plan and announcements for the given course – That way students will have access to all of the latest information concerning the specific course.

Another really good idea that came to our minds which would help us analyze the situation, was to actually ask the students themselves. What a better way to get the best results than to ask the people who are familiar with the Studienet, and have been using it for a while now.

We came out with a list of around fifteen to twenty questions, and then took some time to filter through them, in order to select the best, and form a survey. We put together the survey in such a way that it would contain four multiple-choice questions and one open question that can be answered in just a couple of sentences. This would allow for the students to be able to express their feelings towards the idea we are proposing, and even give us more suggestions to what they would benefit from and how to go about it.

The survey was created using an online website called FreeOnlineSurveys *[1]*. It was the perfect choice for that task, because it is extremely simple to use, contains all the necessary tools in order to make the survey in a professional way, and allowed us to share it the way we wanted. The tool also had the ability to extract the data from the results and present it in various visual ways, from which we could choose. It had the ability to produce different types of charts, some of which: column type, line type, pie type, bar type and many more types of charts.

When we composed the real online survey based on the questions we previously went through, we had to choose how we were going to deploy it, so that all of the VIA University College's students would be able to see it. We figured out that our best bet is to post it online in the VIA University's Facebook page. For quite a while now we have been using the VIA Facebook community page, and we know that it is quite active and every member of the VIA institution is a member of that online community as well. This observation got rid of the fact that some people might miss the opportunity to see the survey we have prepared for them.
So it was decided, we posted the survey and set a time frame of seven days in which we would give the students the opportunity to fill it in. Bearing in mind our limited timeframe to develop this project, we figured that seven days was quite enough time to collect the relevant data we needed.

Survey results

At the end of the waiting period, we collected the data with the results from the survey and we saw that the total amount of people that answered was 67. Here are the questions and results for each one of them:

Question number one: How satisfied are you with the current version of Studienet?

The results were showing that the majority of the students found the web version of the system to be somewhat satisfactory.

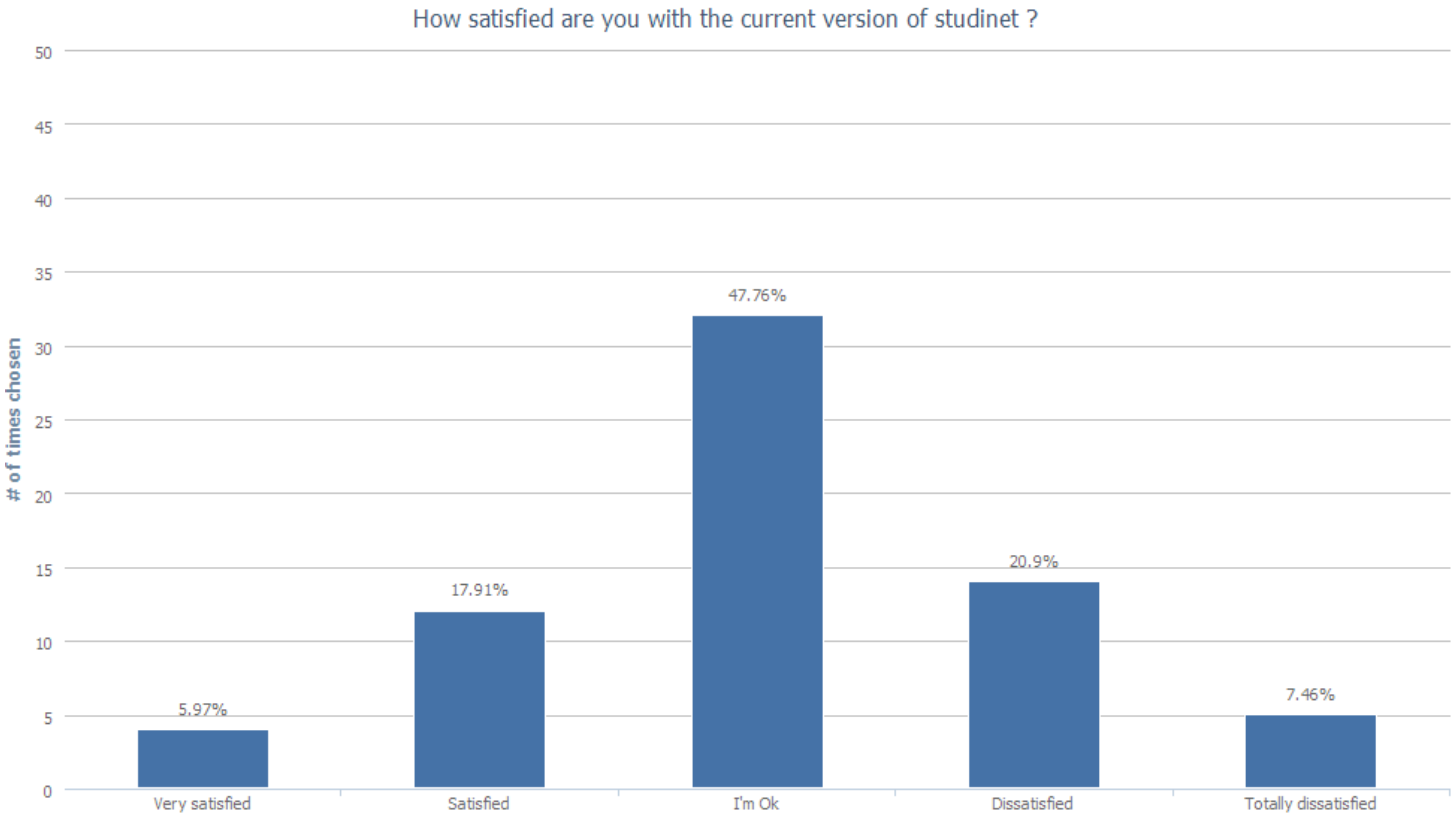


Figure 3 - Survey results for question number one

That alone meant that the web solution was lacking behind at some aspects, and was yet another proof that a mobile solution could fill in the gaps where the web version failed.

Question number two: How do you feel about having a mobile Studienet application in the future?

Here the question is pretty straight forward, and so are the results. It definitely seems that the students of VIA University College are going to appreciate having an application as their personal assistant for the Studienet.

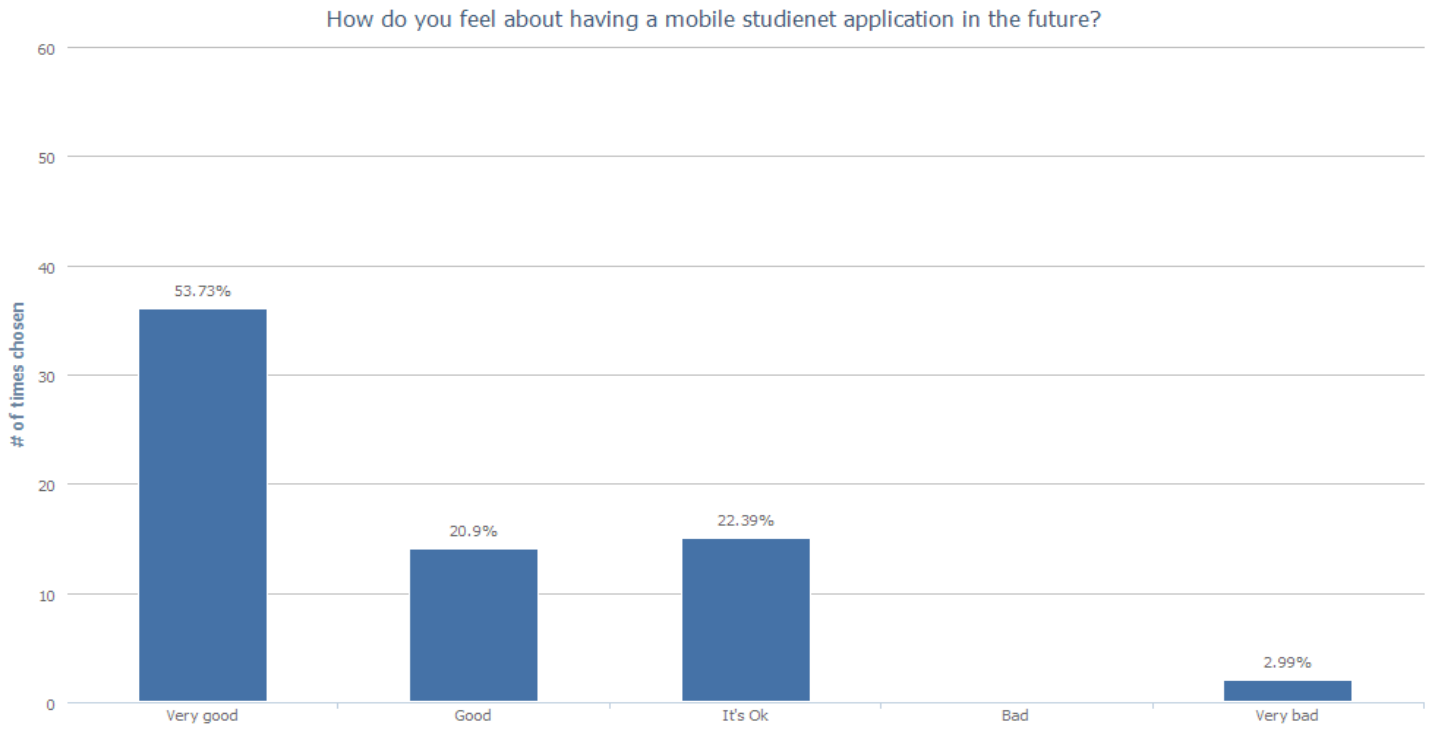


Figure 4 - Survey results for question number two

Question number three: What is your phone's operating system?

Since there are different smartphone operating systems on the market, and our timeframe for the project was limited, we had to focus on only one of them. That is why we needed to know that is the most favorable one throughout the VIA society.

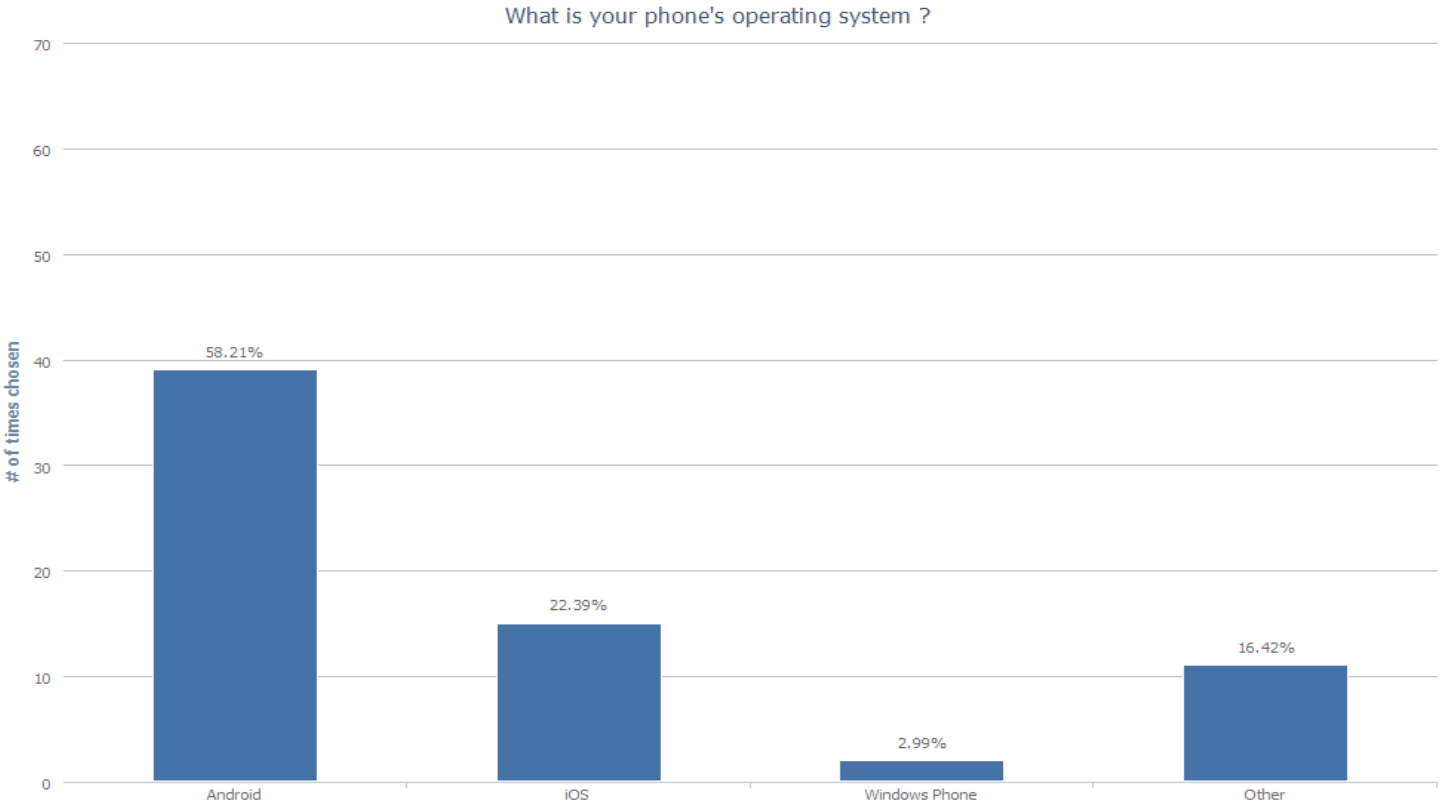


Figure 5 - Survey results for question number three

58% answered in favor of Android, which pointed that this could be our choice of platform.

Question number four: How would you feel about receiving notifications on the application, regarding: e-mails, announcements, deadlines, exam info, etc.?

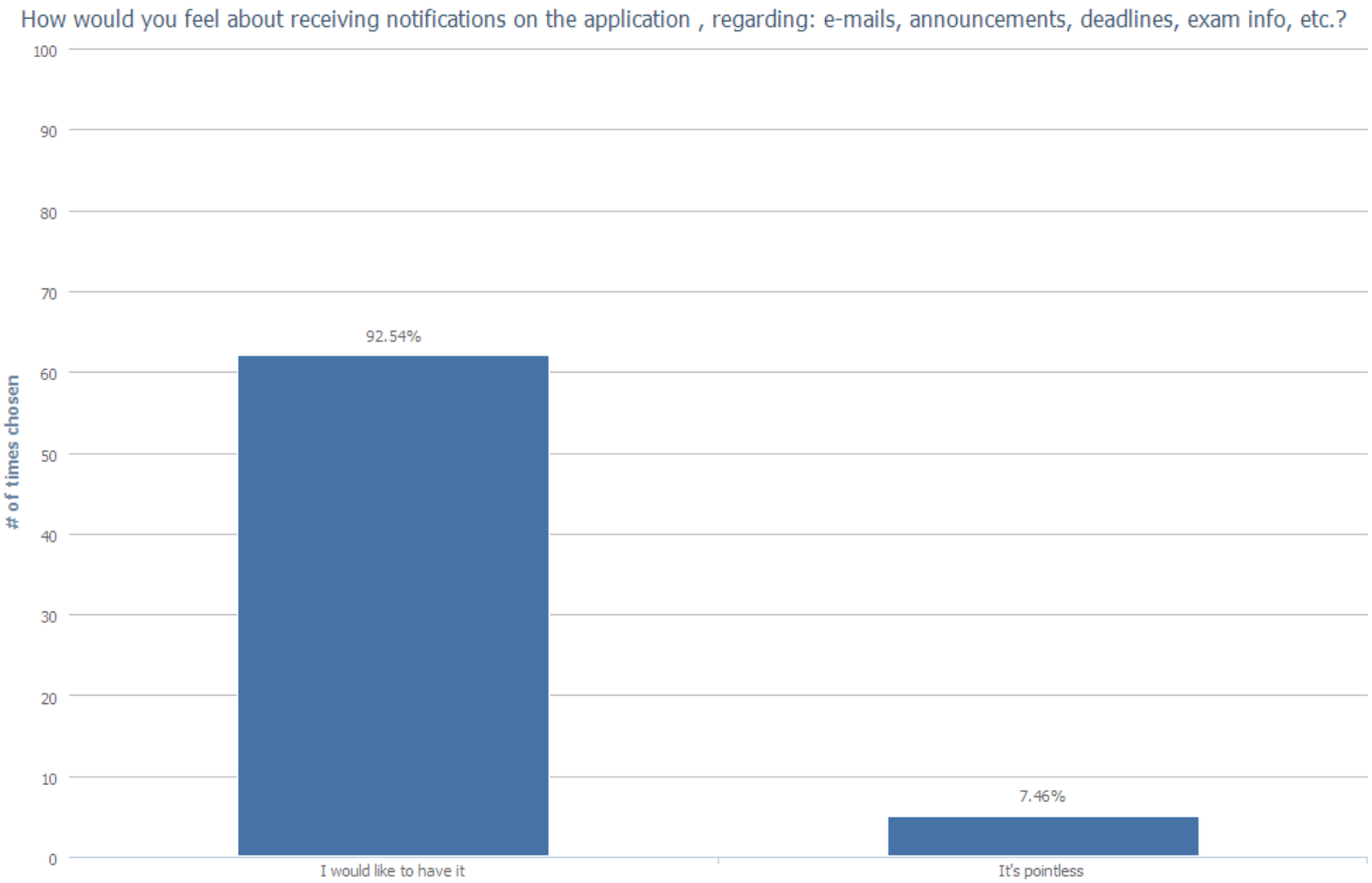


Figure 6 - Survey results for question number four

Here the results are also pretty conclusive, with over 90% of the students who voted, being in favor of receiving updates and notifications on information about VIA and Studienet.

Question number 5: Describe in not more than two sentences, what more would you like to see in the mobile version of Studienet?

Here the students were prompted to respond with an open answer, which could give us some more insight of what more can be added to the system. Because the answers are too many and some of them repetitive or too long, we are going to sum up the main proposals. Here are

some of the answers the students gave us which were different from the ideas that we presented:

- The system needs to be accessible to older versions of Android, so that there are no restrictions to using it.
- A reminder for the events, deadlines, etc. for example one day before the actual event.
- Timetable for classes and the rooms where the classes are going to be held.
- Notifications about canceled lectures.
- Stads log-in availability.
- Access to Studienet materials.
- A fluent and user friendly interface.

To conclude our observations from the survey: we managed to set our sights on a target platform, we managed to confirm that the idea for having a mobile Studienet application is a welcoming thought to the students, and we managed to receive even more good ideas from the them that will contribute to the development of the system.

Technical analytics

For further analysis and development of the Mobile Studienet, our team had to figure out how the current website version operated, and what was the logic behind the scenes. For that to happen we had to contact a person that is either involved in supporting the current version, or either a person that is in charge of management of the Studienet. When we reached out to talk with a representative in charge of the current system, we discussed the opportunity and if we will be granted the permission to operate on the current system. Unfortunately, due to lack of experience on our side, and the system containing a lot of confidential information, we were denied the access to any kind of interaction with the current version of Studienet.

Failing to acquire this vital resource for our development of the mobile application, however did not mean that we were left clueless about the situation.

To solve the issue, our team had a meeting with the team supervisor for this project. We had a discussion and agreed that we have to implement our own version of the system. This version of Studienet, had to be as close to the real one as possible, in order to suffice our intentions of creating the needed functionality.
The only way for that to happen was to try and reverse engineer *[2]* the current system, only by looking at its exposed part to the user, namely the user interface.

Reverse engineering is not an easy task, because during the process a lot of information could be misinterpreted, or even omitted by the person who is doing it. The technique for implementing such a tactic is very hard, and should be treated with great amounts of attention and open mindedness.
After getting the task in hand, our team went very slowly through every scenario a student could experience when using the Studienet website. We created a list of all things a student has access to, all information he or she is allowed to see, and all of the information that he or she is allowed to manipulate.

Here are the main parts of the Studienet website we focused on while reverse engineering it:

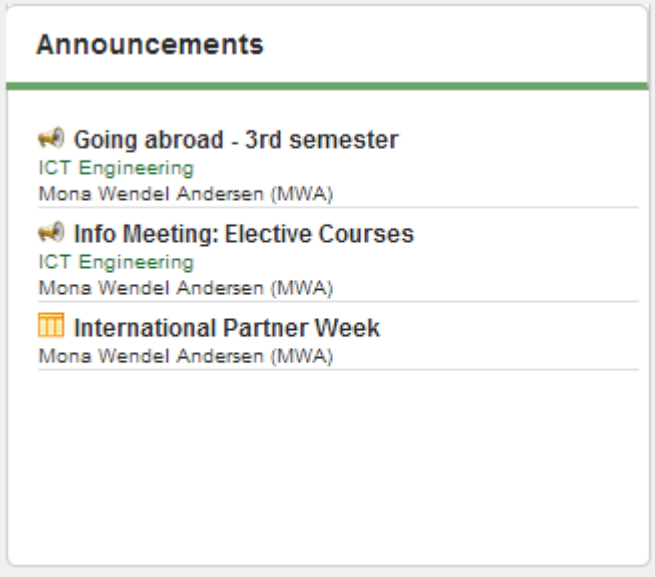


Figure 7 - Program announcements section

The program announcements section of Studienet is the place where every student sees his or hers information concerning the program they are enrolled in. This meant we have to make our backend structure in such a way that it will be able to support every type of information relevant to having the announcement functionality.

The student e-mail along the rest of Studienet's functionality, is one more way for the students to keep track of important information and share it as well. There is no question about it, it is a must to have it incorporated in the mobile system.

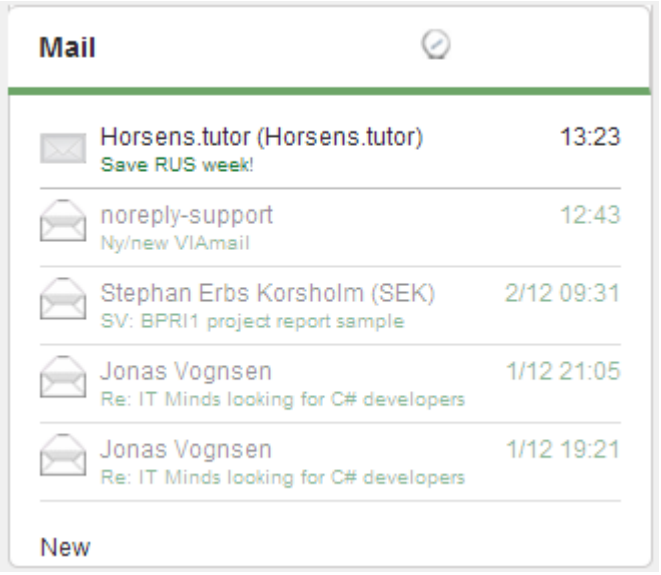


Figure 8 - Email section

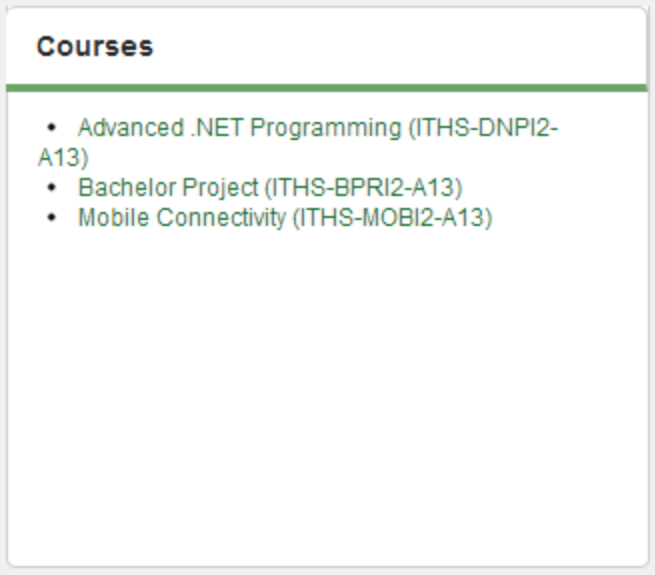


Figure 9 - Courses section

Students are able to see, which courses they are signed up to. That information varies from semester to semester, and from it the students can access specific course details and information about a certain course. In our solution each course is stored along with its information, so that students can access it the same way they do when using the web version of Studienet.

Part of the course's information that the students are able to see when they select a course, are the course announcements. Here each teacher posts information that is relevant to the course. We made sure that not only we support that information, but to include it to the list of features that the user is going to be notified for, when there is a change or a new addition to it.

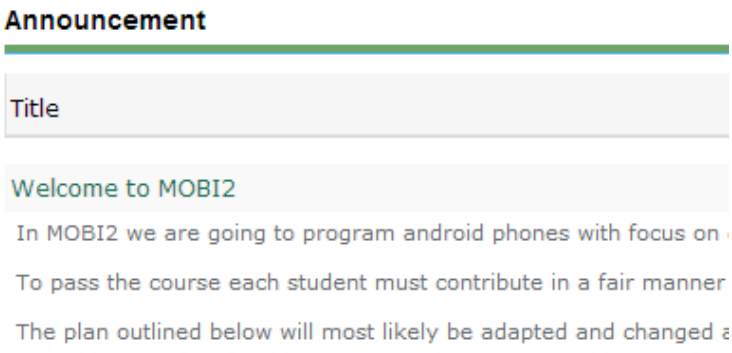

Announcement
Title
Welcome to MOBI2
In MOBI2 we are going to program android phones with focus on
To pass the course each student must contribute in a fair manner
The plan outlined below will most likely be adapted and changed

Figure 10 - Course announcements section

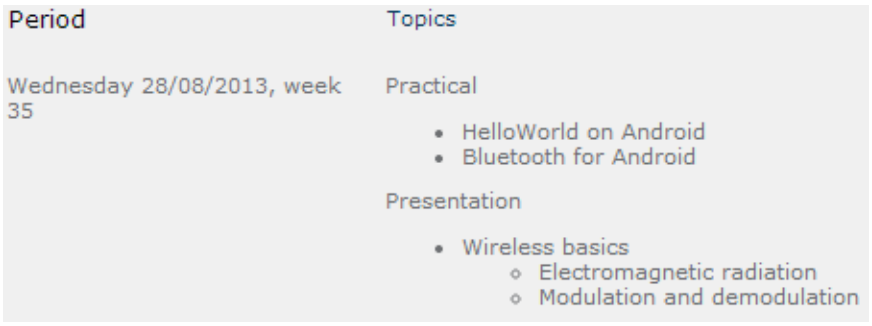


Figure 11 - Session plan section

Another type of information each course has, is the material that is going to be covered in the upcoming lectures by the teacher. This is vital for the students to know in order to be always prepared for their studies.

Figure 12 - Session material section

And last but not least comes in the session material section. Here is the place where all of the teachers' presentations, exercises, examples, lectures and guides are uploaded. From here every student can download and view the things they need.

Basing our conclusions on the survey and the analysis of the web version of the Studienet system, we have prioritized our tasks and combined them into one big use case diagram depicting all of the scenarios a user will be able to do with the application.

Requirements and use case description

This use case diagram specifies all of the requirements to the project and describes all of the actions that will be available to a user in order for him to operate the Mobile Studienet system.

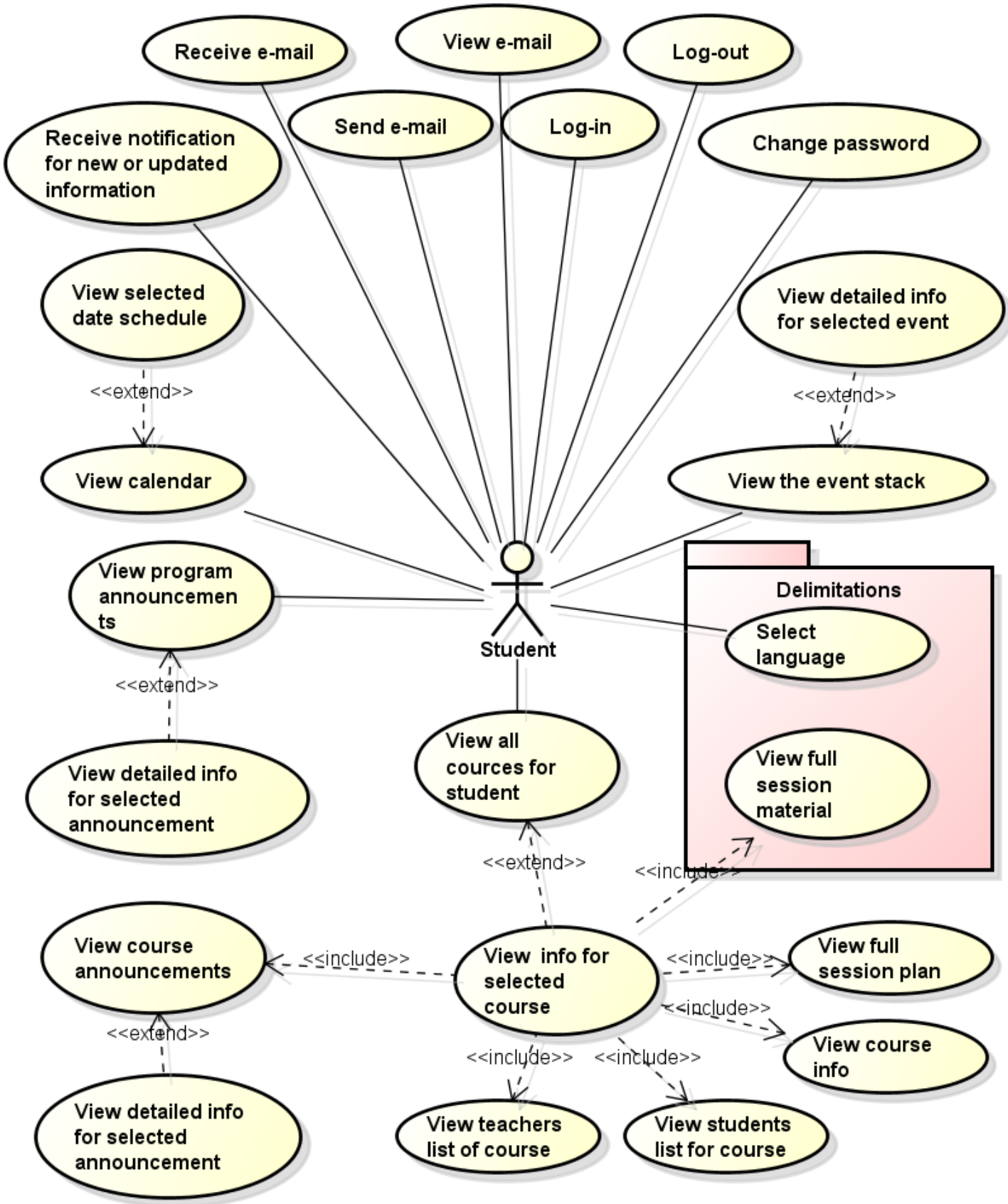


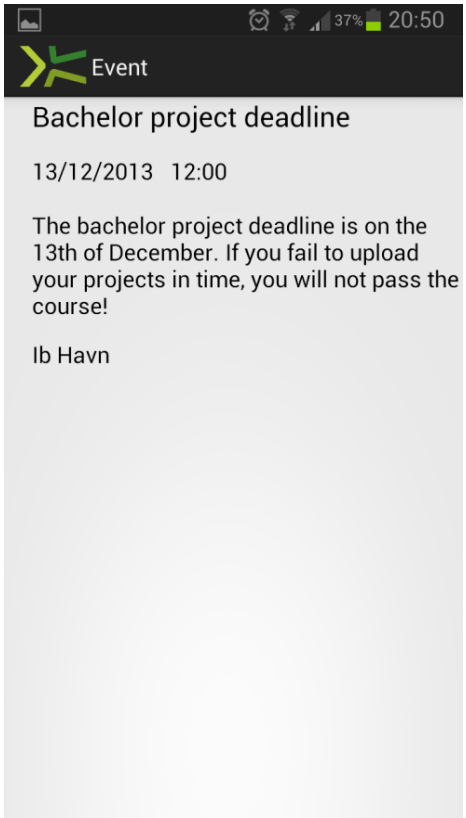
Figure 13 - Mobile Studienet use case diagram

Descriptions of the most important use cases follow in the sections below.

Events

- After logging-in the application, the user is going to be presented to the “Event stack”. The Event stack is the first screen that would be shown when the application is opened, because it is the most important based on our priorities. It is because the events section will hold the most important information for the students, and it is the most important feature the Mobile Studienet has. The Event stack will have all of the events listed by the order of their due date, which means the closest events to come will be on top of the list for the students to notice.

Figure 14 - The Event sack screen of the application


20:50
Event
Bachelor project deadline
13/12/2013 12:00
The bachelor project deadline is on the 13th of December. If you fail to upload your projects in time, you will not pass the course!
Ib Havn

- Another thing is that every event is going to be selectable, and when a user selects it he will be presented with a new screen that will show all of the event information, along with the date and time and the person that made this event.

Figure 15 - Detailed event information screen

Notifications

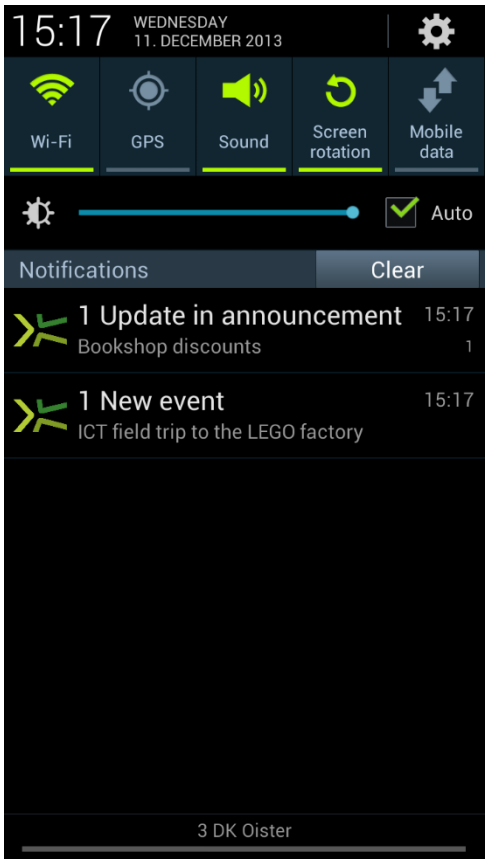


Figure 17 - Notifications of the Mobile Studienet application

- All users will be kept in sync with the latest information, by notifications from the application. They will receive sound and visual represented notifications on their phone, about events, announcements, e-mails, and changes in schedule. This will cover, when something has been added or changed, no matter what type of information it is. When a student selects a notification, he will be directly transferred into the application and shown the details of the notification, so he or she can see the information right away.

Emails

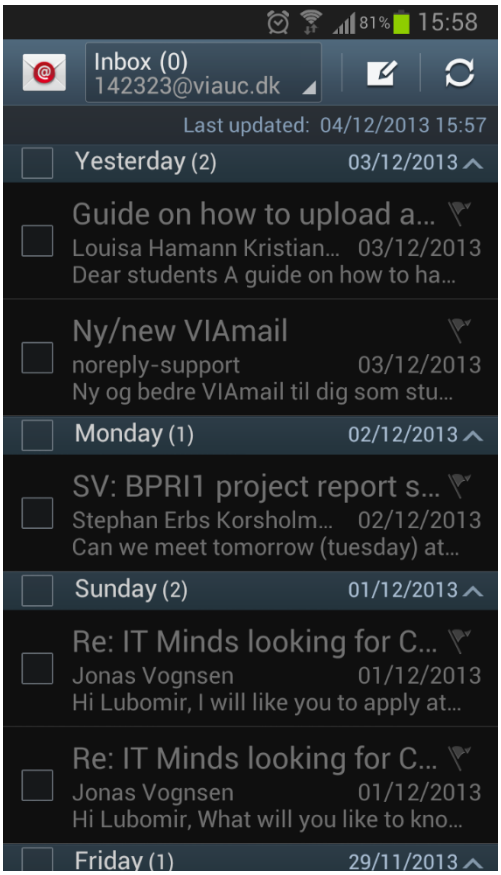


Figure 16 - Android email client synchronized with VIA mail

- The user will have the option to send receive and read e-mails from his or hers VIA e-mail account.
  An investigation we made, led us to the results that Android has a built in email client which can support exactly the type of email that the VIA email is (Microsoft Exchange). That was a very good finding, because in that case we didn't have to build an email client from scratch. Instead the students can set up their VIA email in the default Android email client and everything is done as easy as that.

Program announcements

Figure 18 - Program announcements section of the Mobile Studienet

- A list of all program announcements will also be available, depending in which program is the student participating. When an announcement is available, the user will also be notified about it with notification, just like for the events. And of course detailed information will be available when the user selects a specific program announcement, also like in the events section.

Calendar and schedule

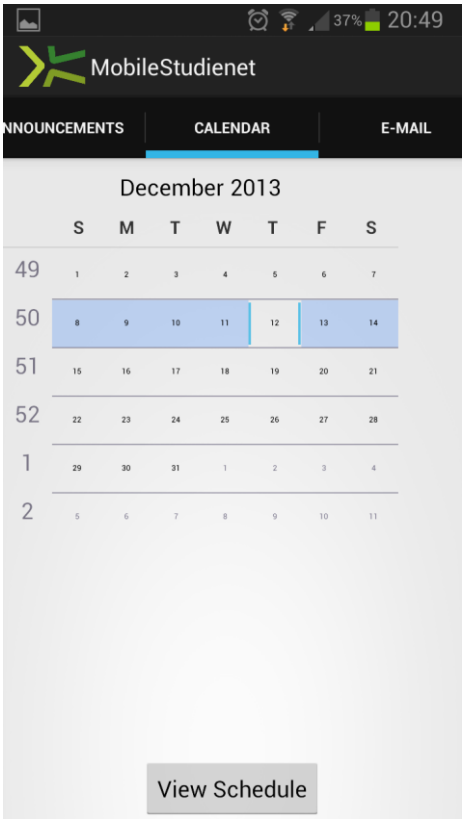


Figure 19 - Calendar screen

- The calendar screen will provide the user with information about their classes' agenda and will show on which days the student has lectures.

Figure 20 - Detailed daily schedule

- Every date square on the calendar will be selectable. When a student clicks on a specific date, a new screen will pop up and will show the day's agenda for all lectures of every course the student is signed up for. It will be represented as a list of the names of the courses and the time frame with the start and end time of each lecture, illustrating also the breaks between lectures.

Courses and course information

- Another piece of information that will be available, is all the courses that the user is enlisted for. After selecting a specific course, the user will be able to see the course announcements, course description, session plan, teachers and students list for it.

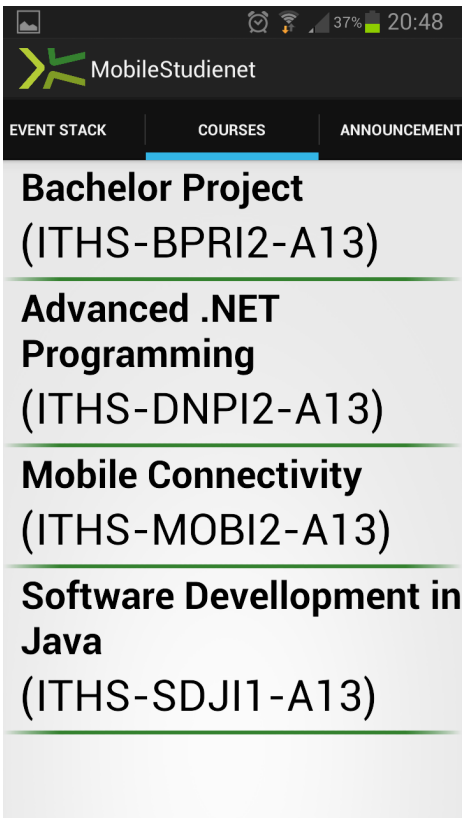


Figure 21 - The screen showing all the courses the student participates in

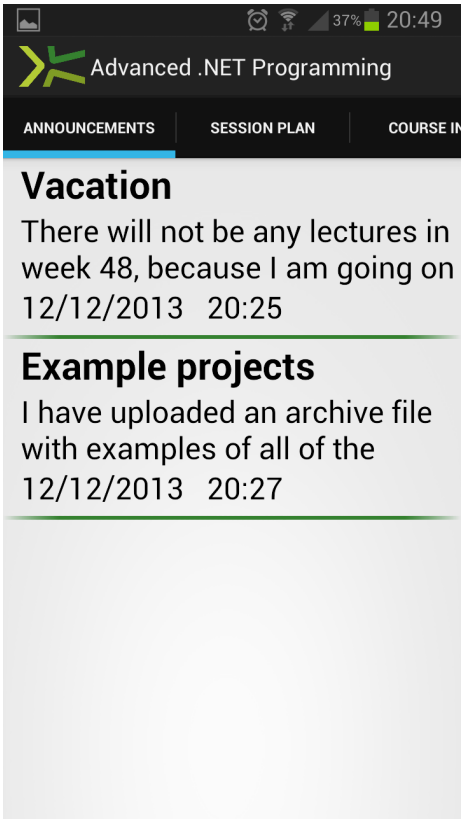


Figure 23 - Course announcements screen

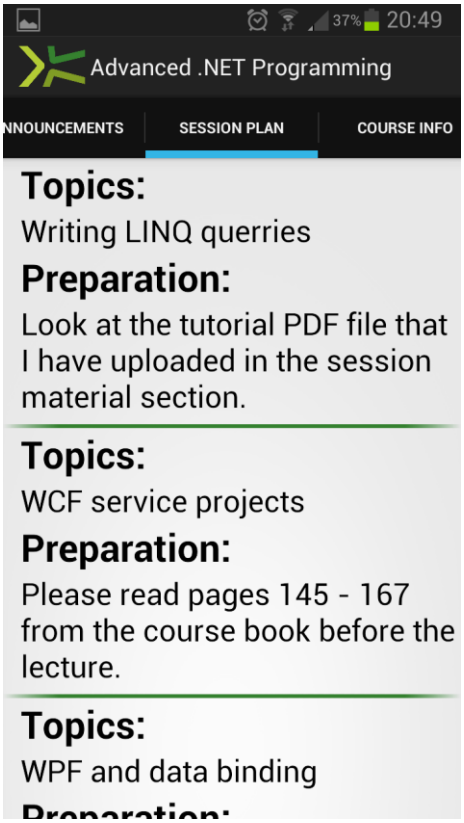


Figure 22 - Session plan screen

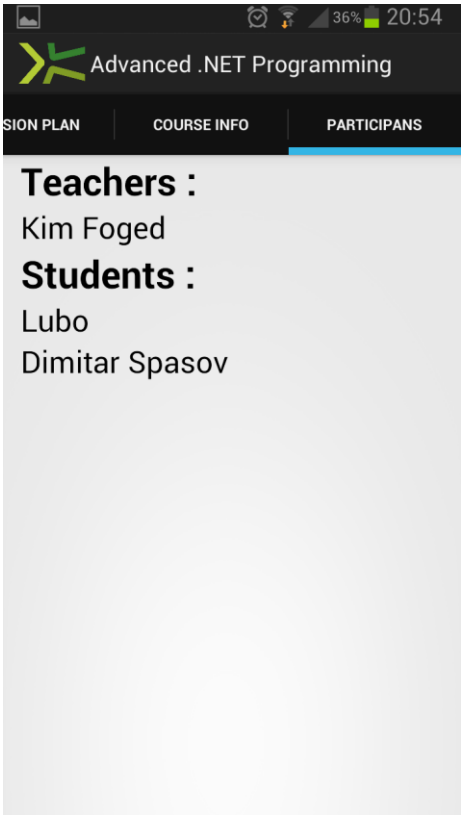


Figure 25 - Screen enlisting all teachers and students participating in the course

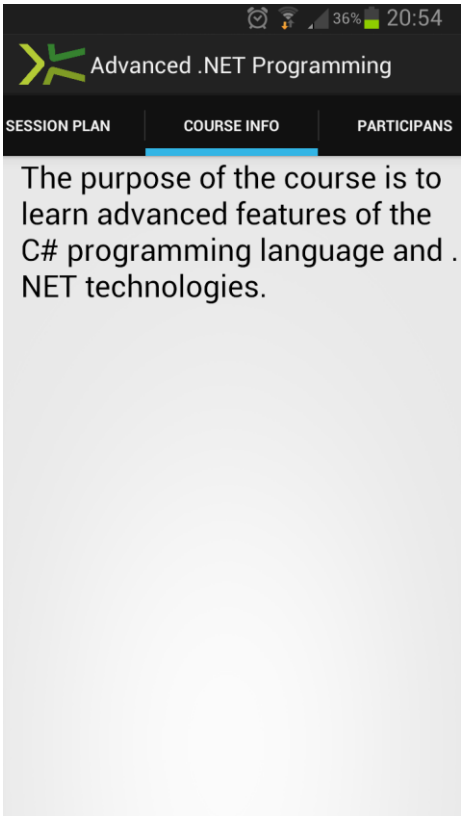

36% 20:54
Advanced .NET Programming
SESSION PLAN
COURSE INFO
PARTICIPANS
The purpose of the course is to learn advanced features of the C# programming language and .NET technologies.

Figure 24 - Course info screen

A more technical description of the use cases can be found in appendix 2 of this document.

This small state chart diagram shows what the map of the Mobile Studienet is and what the user can do:

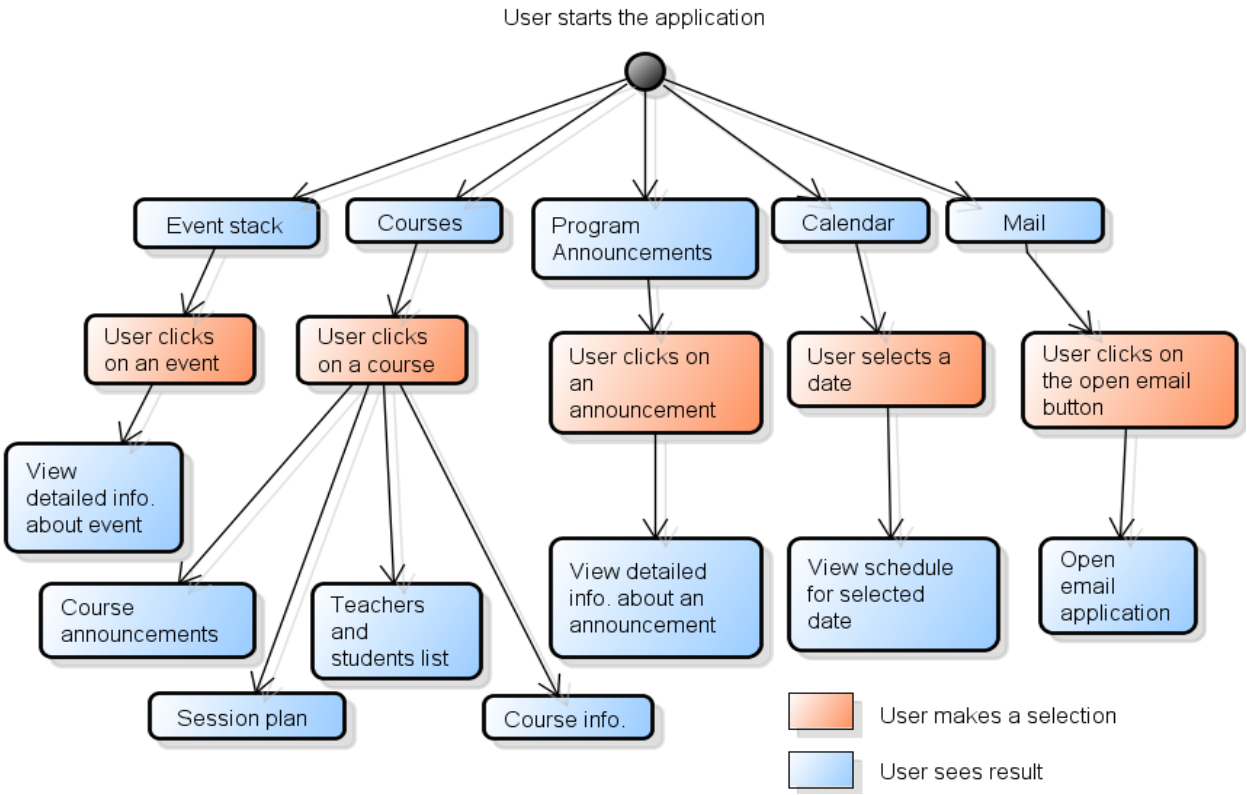


Figure 26 - State chart map of the Mobile Studienet system

This abstract ER model diagram describes in short what some of the most important database tables look like with their main fields.

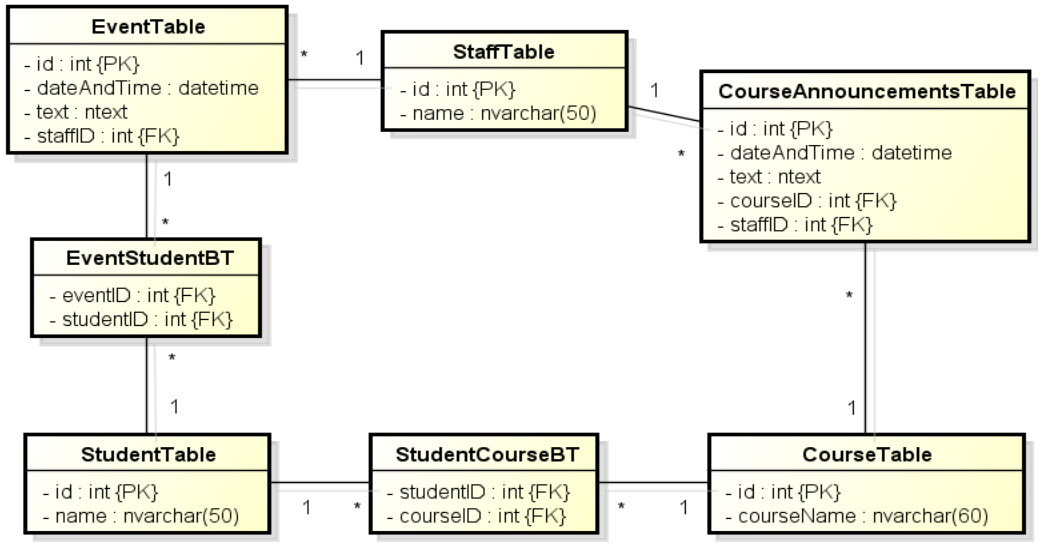


Figure 27 - Abstract ER model diagram of the database

Platform availability

Even though we had our minds almost set on this topic by looking at the answers to question three from the survey, we just couldn't help but to investigate a bit further. There are three major platforms on the market right now, and they are: Android, iOS and Windows Phone. Those are the operating systems almost all smartphones use nowadays. To select one we made a list of the advantages and disadvantages for each one.

- Android – Created by Google, it has the largest market share of the smartphone industry in the entire world, which means it is the most common to use platform out there. Also I is the cheapest platform to distribute, which means that a wider audience of people will be able to afford a smartphone with Android.

- iOS – Created by Apple, is the second most popular mobile platform in the world and also has a very large market share over the smartphone market. The downside to that option is that in order to even start developing an application for that operating system, we have to have a certified developer license, which costs $100 per year. It is not even allowed to start creating an application if we do not have that license.

- Windows Phone – Created by Microsoft, is the third most popular platform on the market. It is fairly new and still in its baby steps towards success. That's why it does not have as many users out there as the Android and iOS platforms.

Being impossible to develop an application for iOS unless we have paid the $100 fee for a developer license, made us fall back from that option. And to Windows Phone, it is just not that popular and very few people own a device that is running Microsoft's operating system.

The answers from the students to question three, and the outcome from our research were more than enough for us to select Android as the platform of choice for the Mobile Studienet.

User experience

After deciding on a platform and exploring all the necessary functionality our system should have, we began to investigate what would be the best way to create a satisfactory and light user experience that the students are going to like.

We analyzed the situation and bearing in mind that the application will be used by people from different spheres of education, it has to be intuitive and organized in such a way, so that everyone can find their way around when using it.
This was the time when one of our team members gave a very interesting idea. He noted that every Android powered device has the Google Play Store *[3]* application. This means that every user that has an Android device, is more than accustomed to the store. And he proposed that we should use a user interface design similar to that of the Google Play Store. That way everyone that will use the Mobile Studienet will know how to use it from the very

start, making it very user friendly and comfortable.
Such a good idea just cannot be missed, and the team immediately agreed on this.

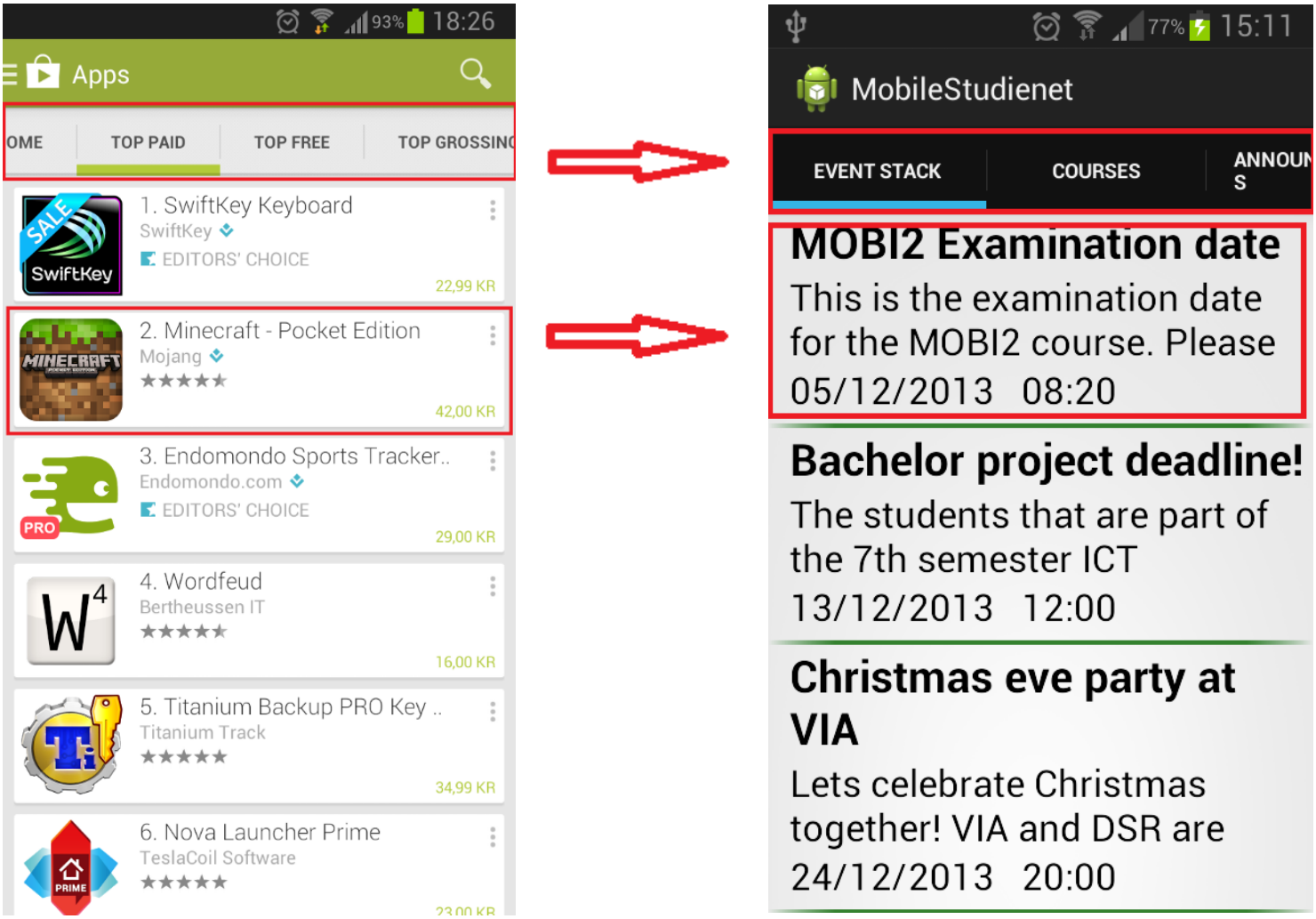


Figure 28 - User interface comparison illustration between the Google Play Store and the Mobile studienet applications

This illustration describes how we got inspired by the design of the Google Play Store, and how we incorporated the same user interface into our application. Making the design of the Mobile Studienet in such a way, makes it look nicer, cleaner and makes navigation between sections seamless.

Delimitations

- For now our application will not support the Stads access for the students. Stads is the Danish student database, where all grades and all information for all students is stored. The reason for this is that, Stads contains very sensitive information about all students and their credentials, and thus we leave it out as a future option.

- Support for other mobile operating systems, will not be implemented at the current state of the system, as we are very limited in time, and in order to incorporate support for the other OSs, we will need the time for developing this project times the number of platforms.

- Since this university is international, which by itself states the fact that everyone studying in it is capable of speaking English. That for now at least eliminates the urgent need of incorporating multiple language functionality to the system.

- Session materials access will be omitted, due to the fact that file management using a mobile device is not that mature at that time and some of the files might not even be supported for the user to interact with. This made us rethink our priorities concerning the session materials functionality and leave it off the scope for the project for now.

So it was settled! We had the requirements, we had the planned functionality and the platform of choice, we had the user interface decided on, and to add to it all, we had our analysis and research to back things up. Now we know what we need to do and it is time to start incorporating all of these ideas we made earlier to create the Mobile Studienet.

Tools and technologies

This section of the report describes all of the technologies used to create the Mobile Studienet application and re-create the Studienet backend infrastructure. It is also argued between different types of solutions for the different problems we faced, and explained what made us pick the right choices.

The backend

Since we had no idea of what the backend looked like and how it operated behind the scenes, we were forced to observe and analyze the Studienet webpage in order to try and guess what its semantics were. That meant we had to choose all of the technologies to support out version almost blindfolded, and we mean almost, because it wasn't quite like we didn't have any clues to it.

Here is what we managed to conclude based on our observations:

- Being a student at VIA, you are granted a special Microsoft DreamSpark account. This account grants students access to paid Microsoft software development tools and material for free. That meant that the VIA University College, must have some kind of professional relationship or contract, with Microsoft representatives.

- Another thing we noted, was that the Email solution that VIA uses for their students, is a web based version of Microsoft Outlook. This was yet another thing to point at the fact that the VIA institution is closely related to the Microsoft Corporation.

- And the final thing we noticed, which didn't take much time to figure out, was to look at the web page URL link. This by itself shouldn't mean much, but in our case it was the proof that converted our assumptions to facts.

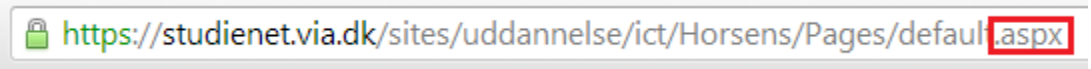


Figure 29 - URL link of the web version of Studienet

The suffix "**.aspx**" to the link you see there on figure 29 actually represents the format of a webpage developed in ASP.NET *[5]*. It is a web technology that is developed by Microsoft. This observation proves that Studienet is indeed using Microsoft technologies.

From these observations, our team reached the conclusion that we should use Microsoft technologies when we develop the backend server for the Mobile Studienet.

Development tools

1. C# programming language

The C# programming language is Microsoft oriented and is supported by all Microsoft technologies out there. It is a very easy to learn and user friendly programming language. It is very good for creating all kinds of Microsoft related web based solutions, and at the same time to manage databases and process data. That's why it was the best choice in our case.

2. Visual Studio 2012

Visual Studio 2012 is an official Microsoft product and was our choice for the backend development environment. It is mainly used by C# developers worldwide, and it was included in our DreamSpark student accounts as a free download option. The Visual

Studio supports creation of various types of projects, some of which are: web sites, databases, user interfaces and many more. It is perfect, because it has all the functionality we need in one place. That would make our work go faster and smoother. Of course there were other tools that would allow us to develop in the C# programming language, but they are created by third party companies, and do not have all of the things we need in one place. We would have to install different add-ons to the tools and that would only complicate our development.

3. Microsoft SQL Server 2008 and SQL Management Studio

For data storage that will hold all of the information to the Mobile Studienet, we used the Microsoft SQL Server 2008 database. For creating and managing the database, we used the SQL Management Studio. It is a tool that implements developer friendly ways of manipulating SQL Server databases. The Microsoft SQL Server also has a very good integration capabilities with Visual Studio, which furthermore would make our development faster and smoother.

4. Entity Framework

The Entity Framework, is not a tool, but a technology created to simplify the management of databases when they are being used by a program or a website. It works in such a way that it allows developers to link a database to a Visual Studio program, and the Visual Studio handles the connections between the program and the database by itself. It incorporates better security, so that things such as SQL Injection *[7]* do not work when Entity Framework is used to connect a program and a database.

5. WCF Visual Studio project

WCF (Windows Communication Foundation) is a Visual Studio project that is of the web connectivity type. It is almost the same as the ASP project type, but has a bit more features that we are going to require for our project. The main reason why we chose to go with it is that WCF supports multiple transfer protocols, such as HTTP, TCP and UDP, whereas ASP supports only HTTP. This multiple protocol support, would allow us to transfer not only text, but also files. This would come in handy when it is time to incorporate the Session Material section of the Mobile Studienet. Since we have left it out of the scope of the project we will not use these other protocol types, but it is good to have them available for future development of the system.

6. JSON

JSON or JavaScript Object Notation, is a text-based open standard, designed for human readable data interchange. It is derived from the JavaScript scripting programming language, for representing simple data structures or objects. Despite its relationship to JavaScript, it is language and platform independent, which can be used

with different technologies. Since we needed a proper format for the data we send and receive, we decided to use JSON. One more possibility was to use XML, but since we are using a mobile device for receiving or sending the data, JSON is the better choice, because it is more compact. On figure 30 you can see the difference in size of a message sent using JSON and the same message sent using XML.

JSON message:

```
{"JSONDataResult":"You entered: Hello world"}
```

XML message:

```
<XMLDataResponse xmlns="http://tempuri.org/">
  <XMLDataResult>You entered: Hello world</XMLDataResult>
</XMLDataResponse>
```

Figure 30 - Size comparison between JSON message and XML message

When not connected to a WiFi network, users of mobile devices almost always use mobile data connection to connect to the internet on their devices. This mobile data connection always has some kind of restriction to how much data the user can access per month. By using JSON in our application, we substantially decrease the amount of data that needs to be transferred.

7. IIS

IIS (Internet Information Services) is a tool that allows us to publish and distribute our WCF service through the internet, so that devices that use the Mobile Studienet can connect to the database and receive information through that service. IIS is a virtual directory that hosts our service and allows it to be visible to other devices, not just the server that is hosting it. That way no matter where a user is, if they have any sort of internet connection, weather it is through WiFi hotspot or mobile data internet, they can access the Mobile Studienet server and see the information they need.

The frontend

1. JAVA programming language

The JAVA programming language is the native programming language used by all developers around the world for developing Android applications. Do we need any more reasons to use it?

2. Android SDK

   The Android Software Development Kit is the framework that provides a developer with the necessary programming libraries in order for him to develop mobile applications.

3. Eclipse

   Eclipse is an open source development environment that is used mainly for development of JAVA programs. It can also be used to develop in other programming languages, but JAVA is what we are looking for. When it comes to Eclipse, it is the most popular programming environment, and this is because using it is the most convenient way to develop Android software with the JAVA language.

Design and architecture

System architecture

Mobile Studienet has the three tier client-server system architecture, because it is comprised of: a database storing all of the information (tier one), a server hosting the service that can be accessed over the network (tier two) and a client that connects to the server through the network (tier three). It works the same way as the normal Studienet website, but the difference is that at the client side in the mobile version sits a smartphone, whereas at the client side of the web version is a computer loading the web page.

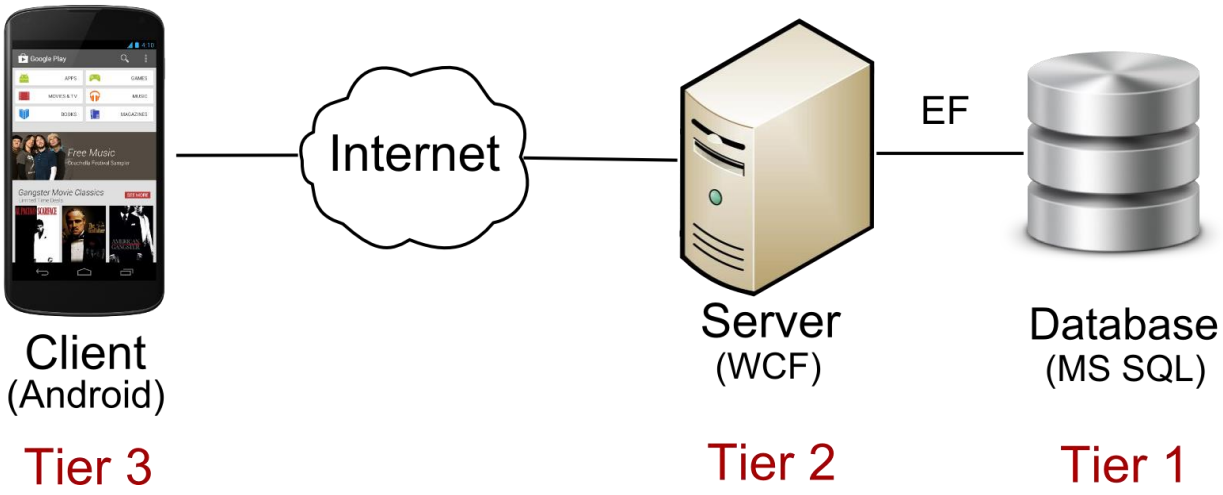


Figure 31 - Mobile Studienet system architecture

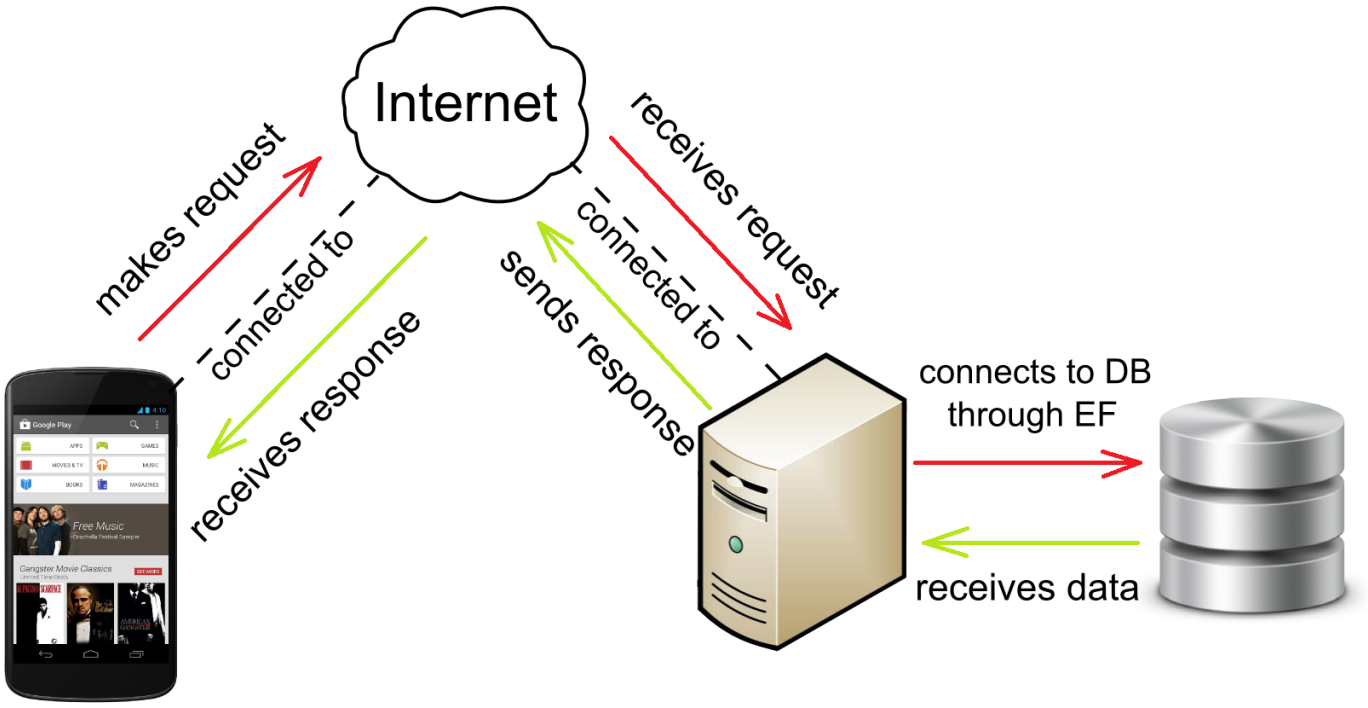


Figure 32 - Mobile Studienet system workflow

The workflow of the Mobile Studienet system goes like this: Whenever there is a change in information or a new information added, the mobile application will send a request to the server to ask for that information. The smartphone makes the request through the WCF web service which is available through the network. The server, which is on the other side of the network, receives that request and processes it. Then it contacts the database through the Entity Framework to ask for the new information. The database queries the data and returns it through the server again through the Entity Framework. Then the server creates a response containing the information and sends it back through the WCF service. When the response reaches the Android smartphone, the application unpacks the response and displays the data to the user.

The backend

The backend of the Mobile Studienet consists of the Server hosting the WCF service and the Microsoft SQL database. In our case right now the server and the database are both installed on the same computer, which means that the second and third tier are combined into one physical server machine.

Database design

The database design and architecture are described using the Entity Relation model diagram seen below.

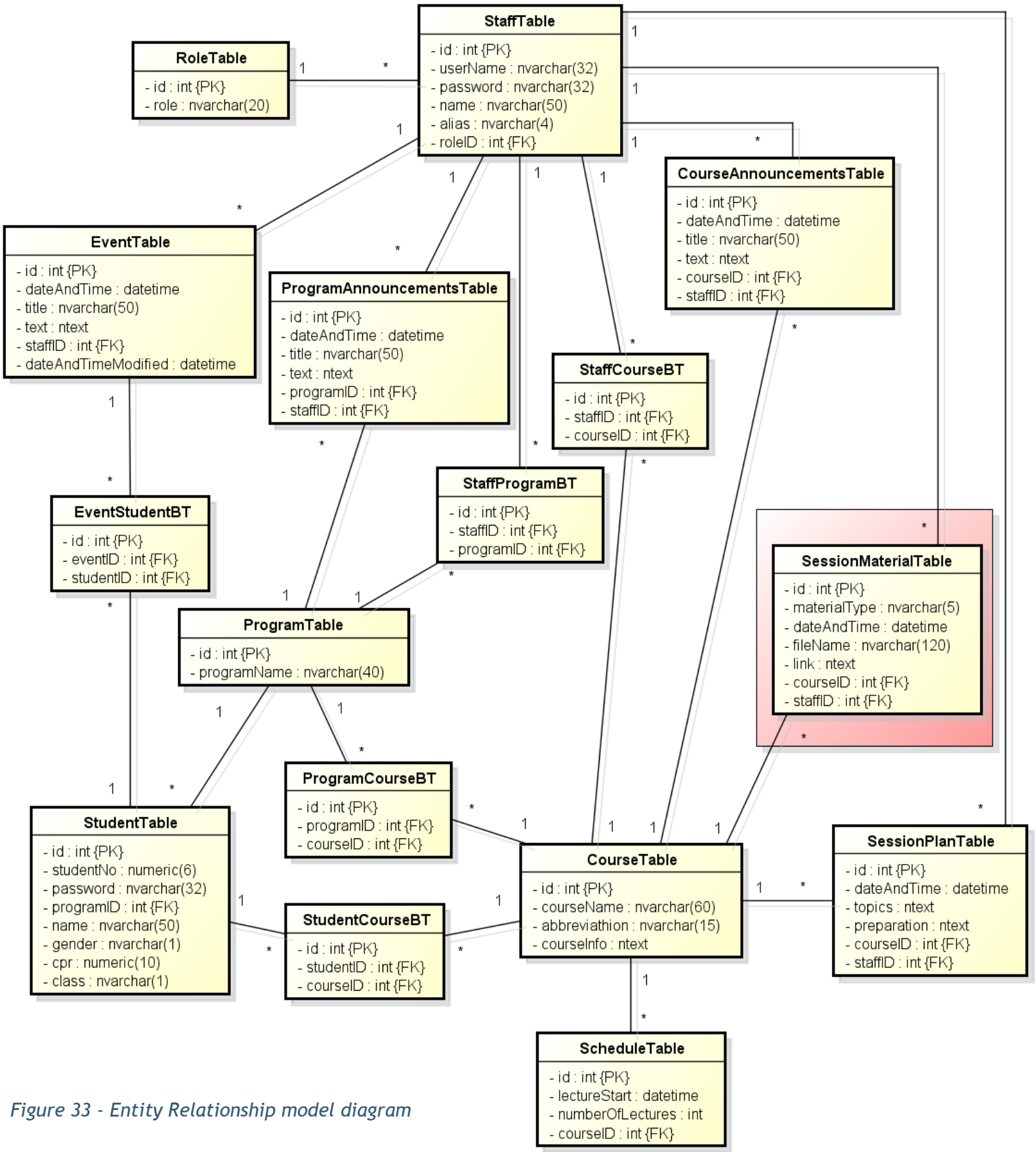


Figure 33 - Entity Relationship model diagram

The RoleTable:

Fields:

- **id** – primary key, it is represented by a uniquely auto generated number.
- **role** – role is the verbal definition of the privilege that is going to be applied to different kinds of personnel that manages the Studienet.

Since the Studienet is a very big system and has a lot of management going on by all kinds of teachers, secretaries, administrators and other personnel, we have created this table which is used to give certain privileges to all staff members.

The StaffTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **userName** – it is the username that every staff member will use to log-in to the system in order to maintain it, or post information.
- **password** – is the password the staff member is going to authenticate themselves with when logging-in.
- **name** – is the actual real life name of the person that this account belongs to.
- **alias** – a three or four letter representation of the person's name, used in the VIA email (i.e. IHA@viauc.dk), or when a staff member uploads some information to the Studienet, and the alias can be seen next to the material they have uploaded like a signature.
- **roleID** – foreign key to the RoleTable, it references the type of privilege this staff member account has.

In general this table holds every information that a staff member may need when using or managing the Studienet system. They will be able to access it with their credentials (username and password), and depending on the role they have been assigned, they will have no or some limitations to what they can do with the system.

The StudentsTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **studentNo** – is the unique student identification number in the VIA Studienet system. It is also used as the student's username when logging-in the system. Also it is used to determine what will be the student's email address for example 142323@viauc.dk.
- **password** – is the second part of the student's credentials that the student will use to log-in the Mobile Studienet.
- **programID** – foreign key to the ProgramTable, it shows which program the student is enrolled in.
- **name** – is the field that holds the students full name.

- **gender** – is a field represented by one character, either “M” or “F” for male or female.
- **cpr** – this is a field holding a number with the length of ten digits, and is intended to hold the students social security number.
- **class** – is a field represented by one character, either “X”, “Y” or “Z” for distinguishing in which class the student is participating.

The students table is made to support every information the Mobile Studienet could need for managing the data about the student. It is based on our reverse engineering observations made earlier.

The ProgramTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **programName** – is the name of the study program for example “ICT Engineering”.

The program is pretty simple, just like the RoleTable. It has only two fields one representing the unique ID of the program entry in the database, and the other is the program itself.

The ProgramAnnouncementsTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **dateAndTime** – it represents the exact date and time the program announcement was created or edited to the Studienet system.
- **title** – this field holds the information concerning the title of the announcement.
- **text** – is the actual body of the text of the announcement. It holds the main information.
- **programID** – foreign key to the ProgramTable, is a reference to which program is this announcement intended for.
- **staffID** – foreign key to the StaffTable, it shows who actually added this announcement.

This is the table in the database that will hold the information for every program announcement entered into the Studienet, for every program. We have made it so that depending which student from which program uses the system, they will only see the announcements relating to the program they are enlisted for. This is done by using the programID reference to see which program does the announcement goes to.

The CourseTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **courseName** – is the name of the course for example “Web and Database Design”.
- **abbreviation** – is the course’s initials that are used sometimes to identify the course, for example ITHS-MOBI2-A13 this is the abbreviation for the Mobile Connectivity (MOBI2) course that is taught in the autumn 2013 hence the A13 suffix.
- **courseInfo** – is the description for every course, what it is going to be about and what is going to be taught during the semester.

The CourseTable contains every vital information about the courses that we managed to extract during the reverse engineering of the original Studienet.

The CourseAnnouncementsTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **dateAndTime** – represents the time the course announcement was entered into the system or the last time it was modified by any staff user.
- **title** – is the field that stores the text that the announcement’s tile is made of.
- **text** – is the field that holds the information that goes into the body of the announcement.
- **courseID** – foreign key to the CourseTable, is the reference to which course the announcement was created for.
- **staffID** – foreign key to the StaffTable, it shows who made the announcement.

The course announcements table is almost identical to the program announcements table, but the main difference here is that this table holds information about announcements that are specific to a courses not to programs. A course announcement will be visible to all students that are enrolled to attending this course.

The ScheduleTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **lectureStart** – is a date and time field, that shows the date of the lecture and the start time of it. Since the lecture times and breaks for a day are constant, from that point on we can calculate the start and end time for all lectures and breaks.
- **numberOfLectures** – is the number of consecutive lectures a course has for the certain date. Since the lectures of a certain course always come right after another, it is safe to assume that the lectureStart and numberOfLectures fields are enough data to store in order to create the necessary schedule for a given date.
- **courseID** – foreign key to the CourseTable, it shows which course does this schedule apply to.

Schedule table is the table to hold all of the data needed to create a viable schedule for every course. It consists of minimum amount of data needed to incorporate a working

scenario of the everyday schedule for every course. As we stated above the lectureStart and numberOfLectures fields are enough to calculate all of the lecture and breaks timeframes and display it to the user in a beautiful and understandable manner.

The SessionPlanTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **dateAndTime** – is the point in time when a session plan has been added or edited in the Studienet system.
- **topics** – it is a field that will hold information about the topics to be taught or discussed in the given lecture.
- **preparation** – holds text about what should the students read or do in order to be prepared for the lecture.
- **courseID** – foreign key to the CourseTable, points at which course is this session plan for.
- **staffID** – foreign key to the StaffTable, shows which staff member uploaded the current session plan.

The SessionPlamTable contains fields for every type of information we found relevant when reverse engineering the website.

The EventsTable:

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **dateAndTime** – this field holds the date and time of when the event should fire. It is a point in time in the future, based on which the application will fire an alarm.
- **title** – is the field that holds the text that goes into the title of the event.
- **text** – is the field that holds the text of the body of the event.
- **staffID** – is the staff member who created the event.
- **dateAndTimeModified** – is the field that stores the date and time of when the event was created or last updated.

The EventsTable along with the two announcements tables, is one of the most important pieces of the Mobile Studienet, because the notifications that our application will provide to the students, will be because of these three tables.

The StaffProgramBT (Bridge Table):

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.

- **staffID** – foreign key to the StaffTable, it shows who of the staff members is managing the program.
- **programID** – foreign key to the ProgramTable, it represents the program that the staff member participates in.

The StaffProgram bridge table is the link we create between the StaffTable and Program tables. This is because, we have the knowledge that many staff members can manage the same program, but at the same time one staff can manage many courses at once.

The StaffCourseBT (Bridge Table):

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **staffID** – foreign key to the StaffTable, it shows who of the staff members is responsible to teach a certain course.
- **courseID** – foreign key represents the course that is taught by the staff member.

The StaffCourse bridge table is the link between the StaffTable and CourseTable. It represents the many to many link, in such a way that one staff member can teach many courses, and at the same time one course can be taught by many staff members.

The ProgramCourseBT (Bridge Table):

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **programID** – foreign key to the ProgramTable, it represents the program that the course is associated to.
- **courseID** – foreign key to the CourseTable, represents the course that belongs to the certain program.

This table depicts the relationship between the ProgramTable and the CourseTable. While a program can have many courses that are taught in it, at the same time a course can be taught by many programs. For example Global Business Engineering students sometimes take courses from the ICT Engineering program.

The StudentCourseBT (Bridge Table):

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.

- **studentID** – foreign key to the StudentTable, it represents the student which is taking lectures in a certain course.
- **courseID** – foreign key to the CourseTable, it shows which course is the student taking lectures in.

The StudentCourse bridge table is the link between the StudentTable and the CourseTable. We made it like that, so it will support the scenarios of one student taking lectures in many courses at the same time, and that one course can be visited by many students at the same time.

The EventStudentBT (Bridge Table):

Fields:

- **id** - primary key, it is represented by a uniquely auto generated number.
- **eventID** – foreign key to the EventTable, it shows which event is valid for which student.
- **studentID** – foreign key to the StudentTable, represents the student that this event applies to.

This table represents the link between the EventTable and StudentTable. It works in such a way that it allows a student to be notified about many events at once, and at the same time one event to be applicable to many students.

The SessionMaterialTable:

At last comes the session material table, its intention was that it will hold all of the presentations, examples and documents the teachers upload to Studienet. The reason it is marked with red, is that we decided to leave out this functionality from the scope of the project, due to insufficient time. It is left in the diagram though, to indicate that it is an object of further improvement to the current state of the Mobile Studienet.

WCF service and Entity Framework

For the web service connection between the client (Android application) and the server, we chose to use a WCF Visual Studio project. We had a couple of other contenders for the job, one of them was an ASP project, and the third one was MVC project. Here are mentioned all of the descriptions of the different projects and the reasons we chose WCF instead of the others.

1. WCF project:

   Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.

2. ASP.NET project

   ASP.NET is a server-side web application framework designed for web development to produce dynamic web pages. It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.

3. MVC project

   Model–view–controller (MVC) is a software pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. The central component, the model, consists of application data, business rules, logic, and functions. A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts input and converts it to commands for the model or view.

The reason we used WCF over ASP, is because as stated above WCF allows for binding to the server using different types of connections at the same time like HTTP, TCP or UDP, whereas ASP allows connections over the HTTP protocol. If we had chosen ASP for the connection, things like file transfer (which uses TCP connection) for the session material data for the Mobile Studienet application, would have been impossible. Even though we haven't implemented the functionality for it, we have to be prepared to implement it in the future.

MVC is a type of project that is comprised of three parts. Information storage part (Model) which stored different types of data that is useful for the user. Information management part (Controller) which manipulates the data that is stored in the Model part. And a part that presents the manipulated data to the user (View). Usually the view is comprised of some type of user interface that the user can interact with, either to view some graphics, enter information into the system, or read information.

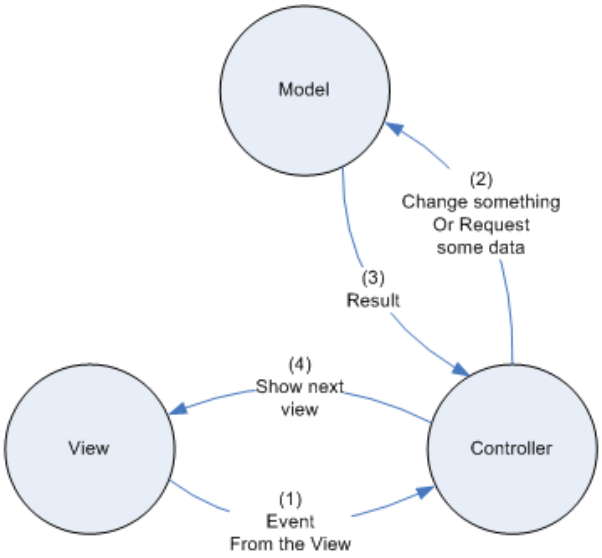


Figure 34 - Model View Controller workflow

Basically the MVC project would have done the same job as the WCF project in our case, because it can support web services, but we did not go with it for one reason only, and that is because it has the View part. Although we have a View that the user can interact with, it is at the Client side of the system, which is in the Android part. A good choice of a MVC project would have been if along the mobile version of Studienet we had to create also the web version. Since the web version already exists and we would want to create only the mobile part, we need only the Model and Controller parts, that's why choosing MVC would have been an overkill.

Figure 35 represents the architecture of our service, and the elements it has inside.

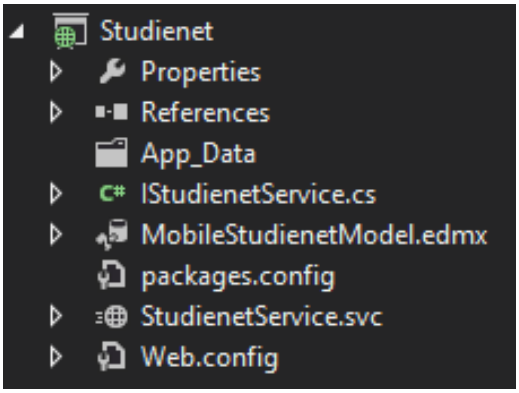


Figure 35 - WCF service architecture

In this WCF project there are three main parts that do all the job of the backend. The first is the IStudienetService.cs interface file. It is the part that is exposed to the network for the client to connect and contains all of the actions that are available to the Android client application. It contains functions that the client can call and then receive results for the requests it made. This interface file is closely connected to the StudienetService.svc file, which is the second of the three parts in the project.

It contains all of the logic for managing the function calls, transforming and saving the information that belongs to our Studienet system. In our case it plays the role of the Controller.

The third, and one of the most important parts is the Entity Framework. It is created and linked to this project. The Entity framework is the link to the database lying underneath the whole system. When we added the Entity Framework to the WCF project, we were asked to point to it, which is our database and where is it located on the server machine. Then based on the tables created in the database, it automatically generated the Entity Framework model classes that correspond to these tables in the database. It created the MobileStudienetModel.edmx file and this file contains all of the Model classes as seen on

figure 36. Every model in the collection, corresponds to one table of the database, which is why there are the same number of model files, as there are tables in the database. There are also a couple of other files in the Entity Framework generated file, but they are responsible for configuration management between the WCF project and the database.

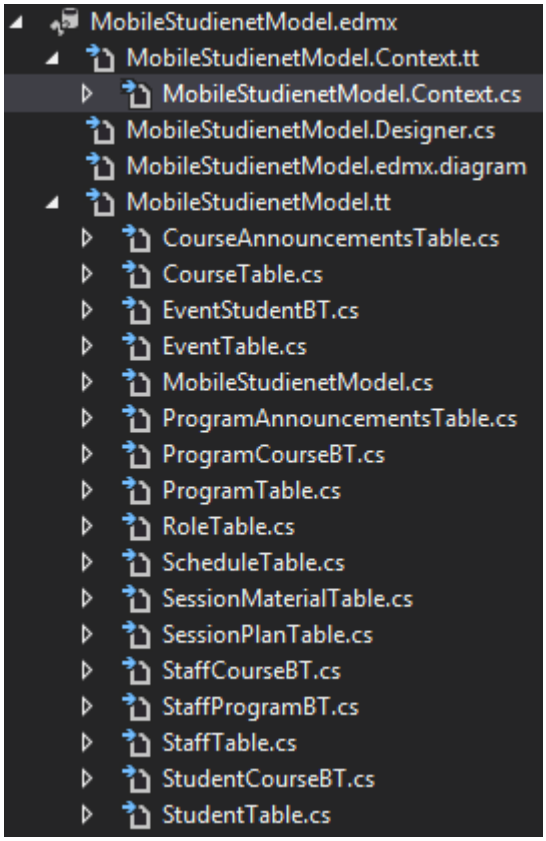


Figure 36 - Entity Framework model classes

Here are some of the advantages of Entity Framework:

1. Reduced development time: the framework provides the core data access capabilities so developers can concentrate on building their application logic.

2. Mappings between the model and the storage-specific database can change without changing the application code. In other words doing something that can cause minor changes to the structure of the database, will not affect the other part of the system, and there will be no need to redesign the code.

3. The applications that use Entity Framework are freed form explicitly typed procedures that use the database. This means that no constant operations are coded in the application in order to access or change the database information. This provides more flexible way for developers to create the programs.

4. It supports Language-Integrated Query (LINQ) support. LINQ is a way to write SQL database-like queries from the code of the program instead using the explicitly typed procedures mentioned in point number 3. While explicitly written SQL procedures, make changes directly to the database, the LINQ makes changes to the model classes created by the Entity Framework, and then these classes apply the changes to the database.

5. Entity Framework provides integrated Visual Studio tools to visually create entity models and to auto-generate models from an existing database (which we did in our case), or can create databases that are hand-edited through Visual Studio for full control.

6. It is well integrated in all of the .NET application projects including ASP.NET, Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF) and others.

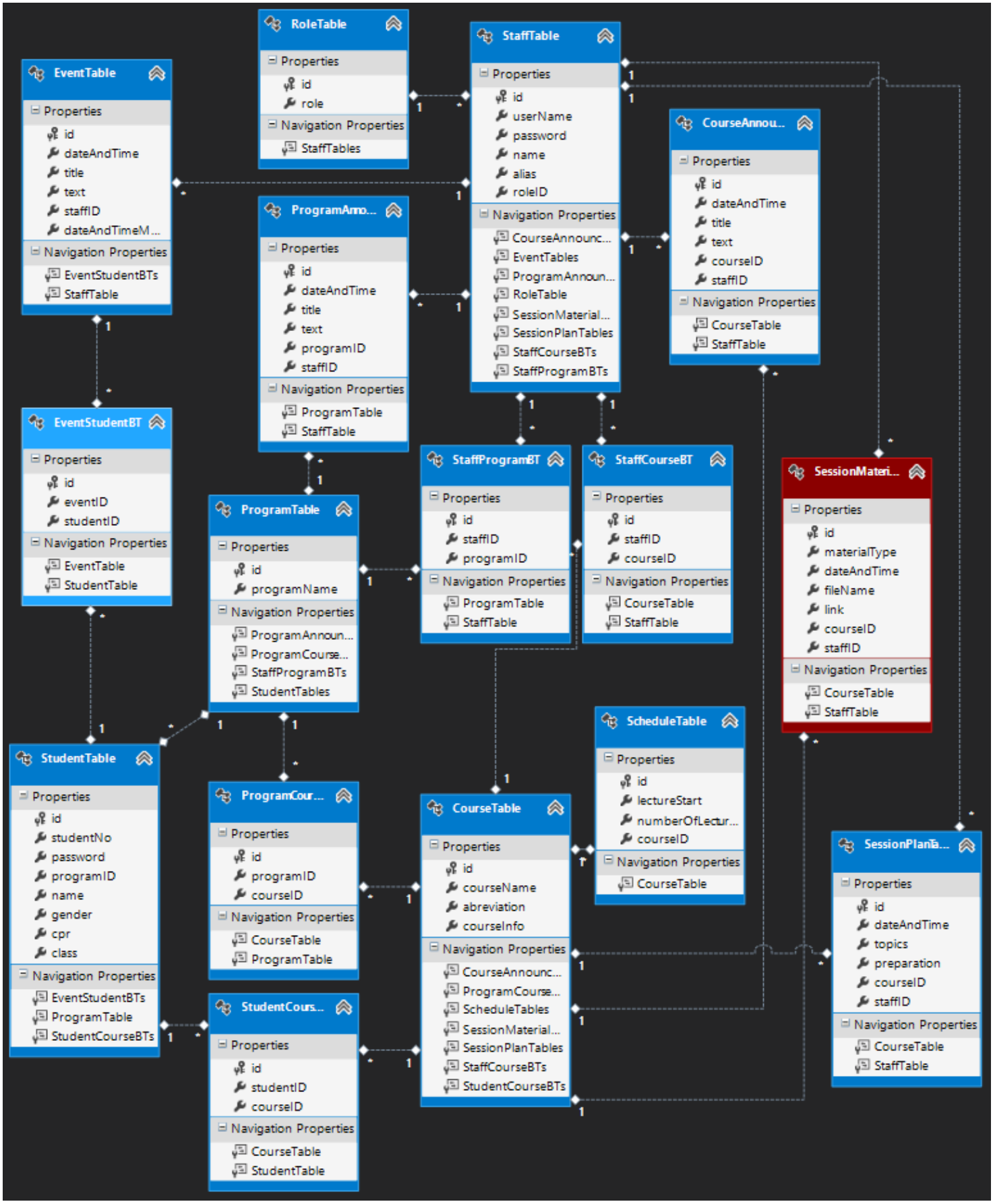


Figure 37 - Entity Framework model diagram

On figure 37 is depicted in great detail what the model classes generated by the Entity Framework look like. If compared to the Entity Relation database diagram introduced earlier in this report, we can see that it looks almost identical, except that this diagram has a bit different information in it bearing in mind that it is describing the model classes, whilst the ER diagram describes database tables.

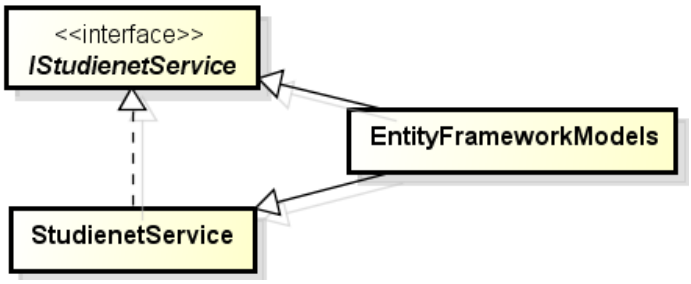


Figure 38 - An abstract class diagram of the WCF service project

Admin Panel

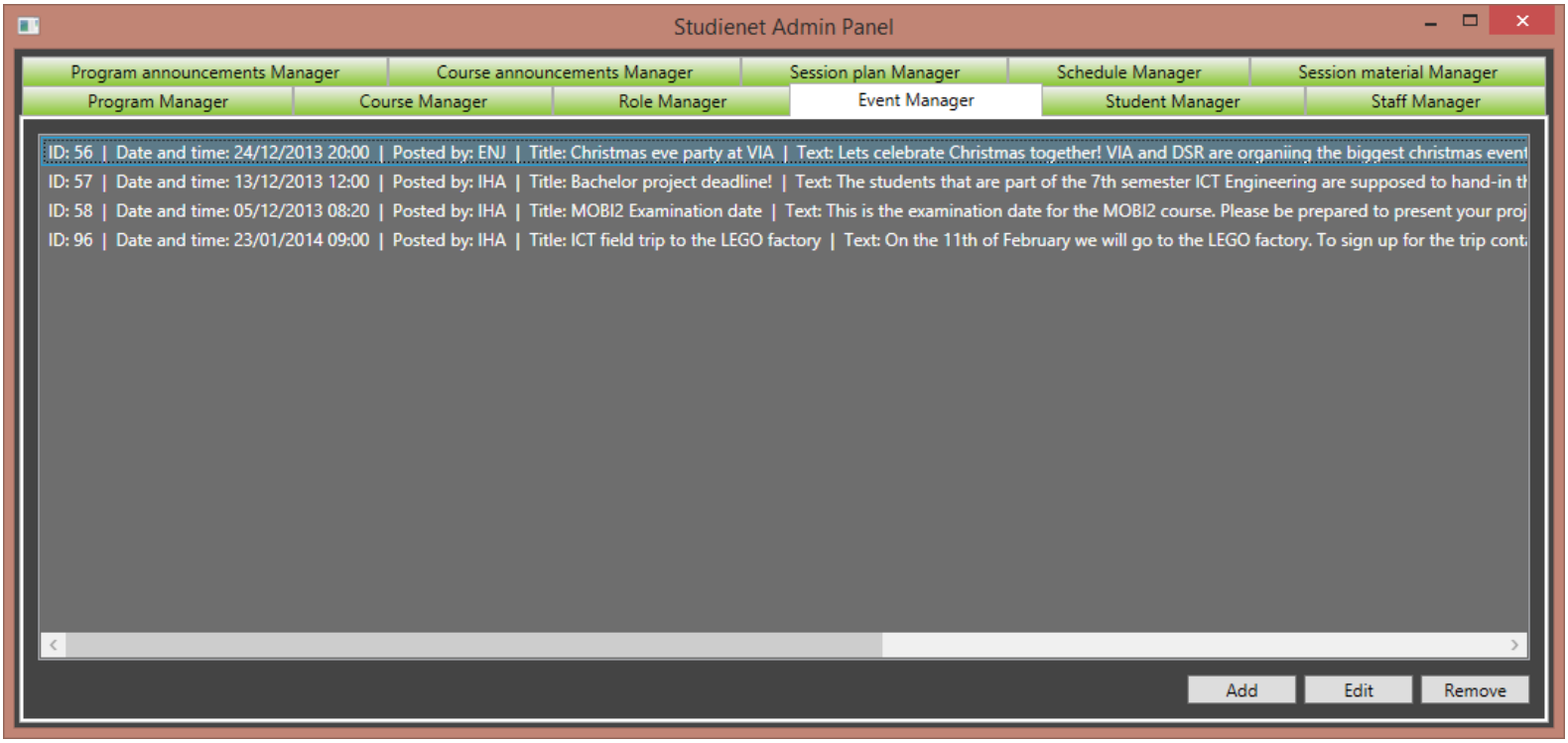

Studienet Admin Panel
Program announcements Manager
Course announcements Manager
Session plan Manager
Schedule Manager
Session material Manager
Program Manager
Course Manager
Role Manager
Event Manager
Student Manager
Staff Manager
ID: 56 | Date and time: 24/12/2013 20:00 | Posted by: ENJ | Title: Christmas eve party at VIA | Text: Lets celebrate Christmas together! VIA and DSR are organiing the biggest christmas event
ID: 57 | Date and time: 13/12/2013 12:00 | Posted by: IHA | Title: Bachelor project deadline! | Text: The students that are part of the 7th semester ICT Engineering are supposed to hand-in th
ID: 58 | Date and time: 05/12/2013 08:20 | Posted by: IHA | Title: MOBI2 Examination date | Text: This is the examination date for the MOBI2 course. Please be prepared to present your proj
ID: 96 | Date and time: 23/01/2014 09:00 | Posted by: IHA | Title: ICT field trip to the LEGO factory | Text: On the 11th of February we will go to the LEGO factory. To sign up for the trip cont
Add
Edit
Remove

Figure 39 - Admin panel - event manager

For easy management of the data that is to be entered and manipulated in the database, we created a very simple program called the Admin Panel (shown on figure 39). We made it as simple as possible as we could, in order to incorporate every CRUD operation for every table that we have in the database. CRUD operations (Create, Read, Update and Delete) are the types of operations that are applicable to any database in order to manipulate its contents. The main reason we created this Admin Panel is just to make the data insertion in the database faster, and save time for implementation of the Mobile Studienet system as a whole.

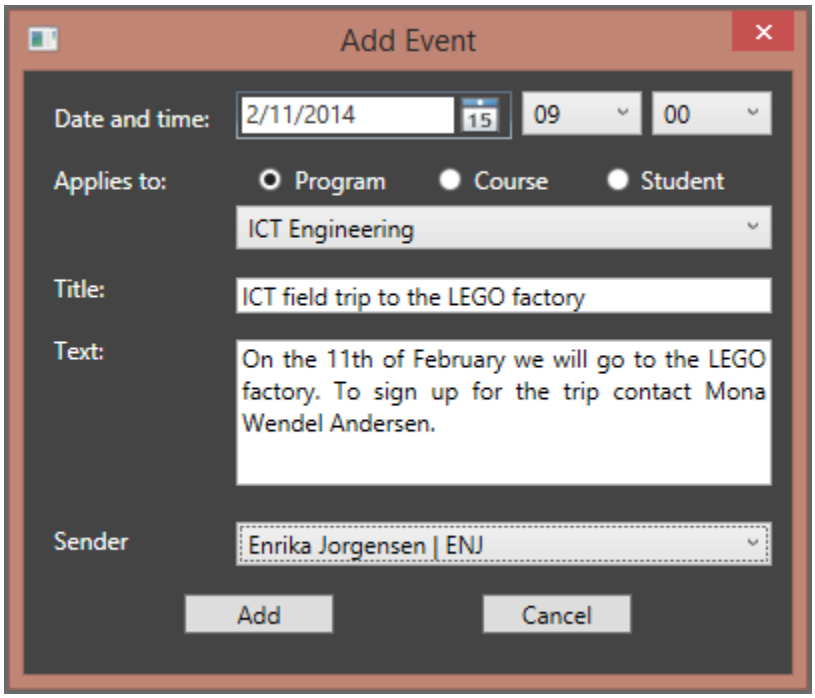


Figure 40 - This is the Add Event window of the Admin Panel program

On figure 40 is depicted the most important feature to the Mobile Studienet system. It shows how events are created. An event has:

- Date and time – this is the point in time when the event is about to take place.

- Applies to radio buttons – these radio buttons allow the wizard to distinguish who the event will be applicable to. It can be applied to a whole program of students, to a specific course that students are participating in, or to a single student, if the event is intended only for one student (for example a re-examination date).

- Title – this field specifies what the title of the event will say.

- Text – is the body of the event where the information about the event is written in detail.

- Sender – the sender field allows the user of the Admin panel to identify him/herself, in order for the students who see the event to know who created the event.

The Admin Panel proved to be very useful when entering dummy data into the database in order to check if things work all right. It also proved very useful when testing the Android application connected properly and if it can receive the correct information. This saved us a lot of time and thus we managed to focus more on the rest of the functionality of the system.

The frontend

The frontend of Mobile Studienet consists of an Android Application Project which is created and implemented in the Eclipse IDE (working environment for JAVA and Android application projects).The Android Application Project is built (installed) on a smartphone device that runs the Android OS. In our case that is a Samsung Galaxy S3.

Android application architecture

The Android application design and architecture are described using the class diagram shown below.

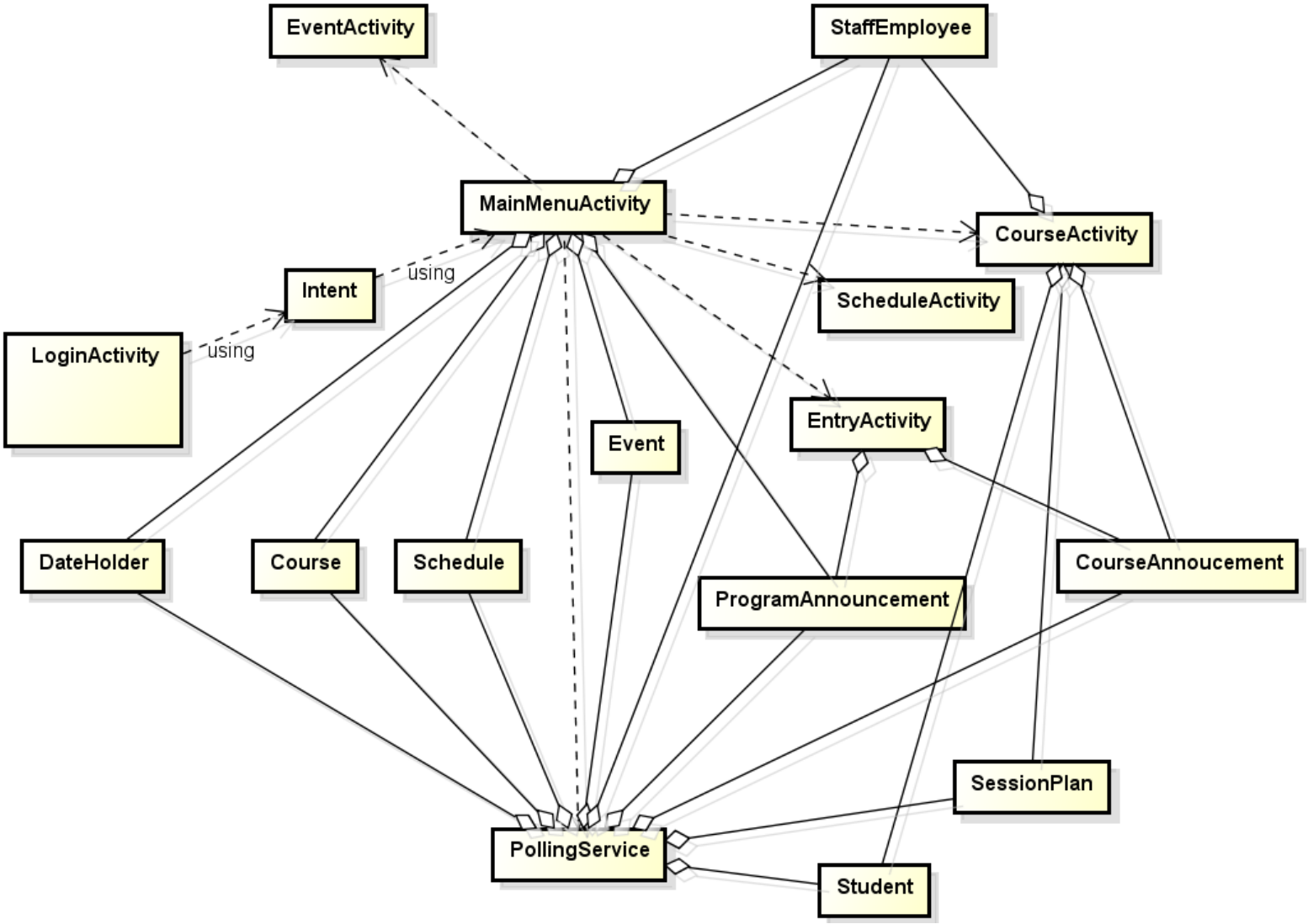


Figure 41 - Android application abstract class diagram

PollingService

In Android a Service is a class from the Application Program Interface, which runs behind the scenes without a direct user interaction (the service has no user interface). Services can continue to run, even if the application is not actively running. This means that the service can continue to do work without being interrupted. Services are used for repeating and long running operations, such as: internet downloads and checking for new data, while working in the background.

Possible ways of fetching data from the backend:

- Push mechanism using GCM (Google Cloud Messaging).

  GCM for Android helps developers with sending data from servers to their Android applications. With GCM, one can send data to his application, only when new data is available, instead of making new requests to the server repetitively (checking for new data every N minutes).

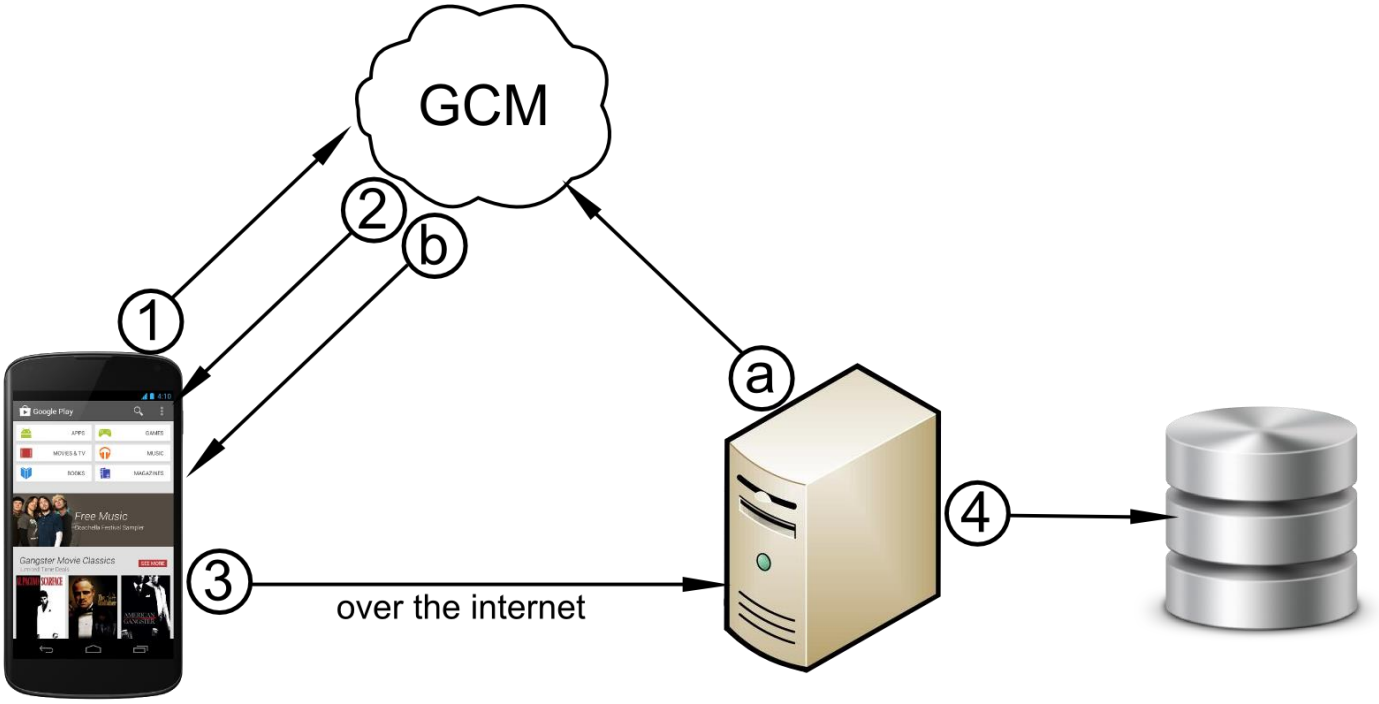


Figure 42 - Google Cloud Messaging workflow

So how does it work?

1. First of all the Android device sends a sender id and an application id (both of which are unique) to the GCM server for registration.
2. Upon successfully registering, the GCM server creates a unique registration id and sends it back to the android device.

3. After the device receives the registration id from the GCM server, it sends that id to the application server.
4. The application server stores the registration id for this device in the database for later use.

    a) Every time a there is some new or changed information in the application server, it will send a message with that information to the GCM server. This message also contains the unique device registration id stored earlier in the database.
    b) Using the unique registration id, the GCM server will deliver the message to the Android device corresponding to that registration id.

- Polling mechanism using an update thread and an Android Service.

  The polling mechanism, simply consists of a thread that is running in the background (in the Android service). This thread will be executed once, every N minutes. Its task is to request data from the application server, and compare the new received data, with the current data stored on the device. If the new data is different from the current data, the user interface of the Android application is updated with the new data.

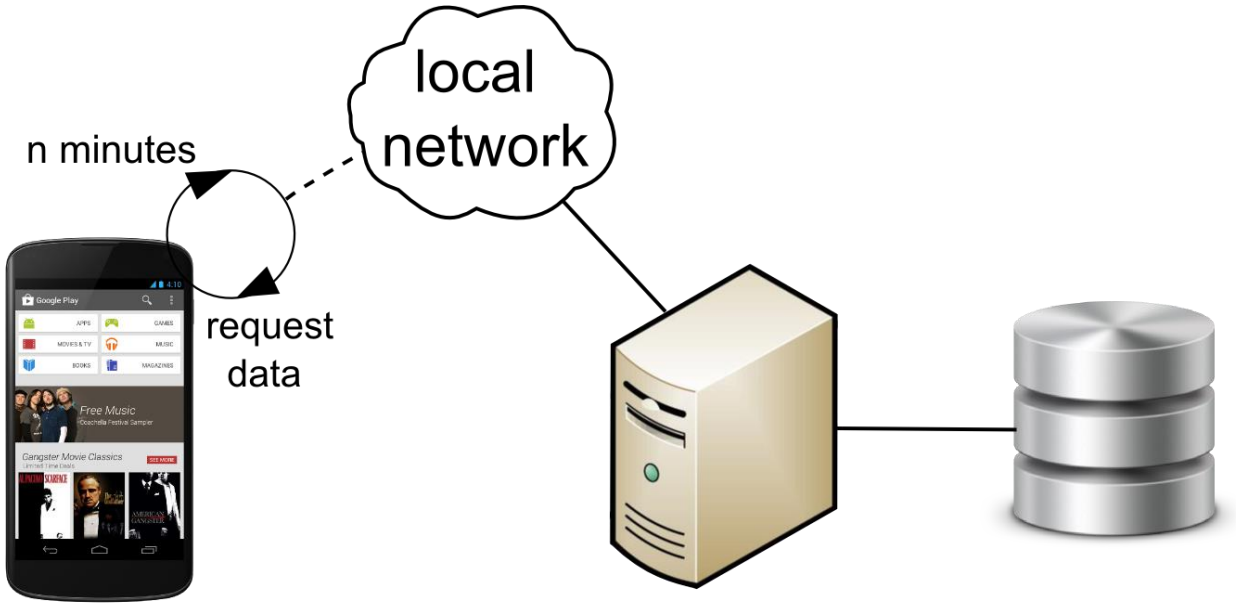


Figure 43 - Polling mechanism workflow

Advantages of using push mechanism over polling mechanism:

- Less load on the application server. The querying is done only when there is a change in the information.
- More accurate updates. The data on the mobile device is updated immediately after the information has been changed on the application server.

- Less battery consumption. Since the mobile device does not have to send requests to the application server every N minutes, the battery consumption is reduced because the processor does less work, compared to when using the polling mechanism.

In our case, the Mobile Studienet application was implemented using the polling mechanism to retrieve the data. This was due to technical issues, which were beyond our control. In order to use the Google Cloud Messaging, we had to have our server exposed to the Internet. However we did not manage to achieve that, because we were unable to configure the local wireless router with the correct settings, due to the lack of the access credentials. We are aware though, that the push mechanism is superior to the polling one, which we were forced to implement in our solution.

The polling mechanism in our system is managed by the PollingService class. Every N minutes, the polling service fetches the data from the backend in a JSON format and unpacks it, so it is readable by the application. The unpacking of the data is done with the help of the model classes: Event, ProgramAnnouncement, CourseAnnouncement, Student, Schedule, StaffEmployee and SessionPlan. Once the unpacking is done, the service compares the new data to the current data on the device. If there are new or updated entries, the service fires up a notification, so the user is made aware of the new information. Next, the service sends the latest data to the **MainMenuActivity**, which updates the user interface of the application.

MainMenuActivity class

Figure 44 - Log-in screen

In order for the user to start using the application, they have to log-in successfully, with a viable VIA student account. Only after a successful log-in, will they be able to enter the main menu of the Mobile Studienet seen on figure 45. The main menu of the application is represented by the **MainMenuActivity** class.

(*In Android every Activity class represents a separate screen in an application.*)

The **MainMenuActivity** class contains references to the user interface sections/tabs, which hold the relevant information for the currently logged-in student. These user interface sections/tabs consist of: Events, Program announcements, Courses, Schedule, Course announcements, Session plan, Course info, and Participants. When new information is received from the server, the **MainMenuActivity** class updates these sections, so that they display the latest data.

Another major role of the **MainMenuActivity** is to manage the data of the application between the other activities. In Android, communication between different activities, happens through an Intent object.

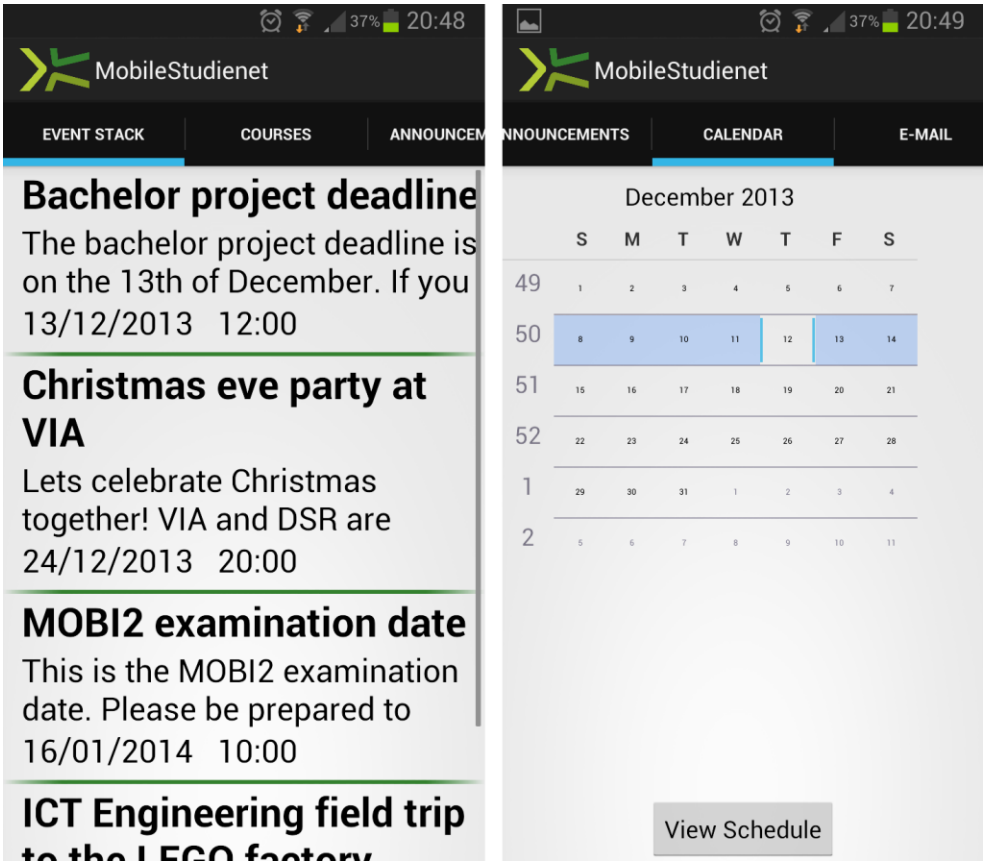


Figure 45 - Mobile Studienet main menu

This intent object is used for starting the other activities (screens). The intent object contains the names of the current activity, the activity that we want to navigate to, and also can contain some data to be passed to the activity we want to go to. Upon a click event (for example, when a user clicks on a specific course from the course list), the new screen starts (the screen corresponding to the clicked course) and gets the Intent object that is sent from the previous activity. From this object, can be extracted all of the information that the second activity might need (for example the course announcements, session plan, course info and participants list). All of the data management for the Intent object, is done in the **MainMenuActivity.** For the sake of making the class diagram more readable, the intent object is displayed only between the **LoginAtivity**, and the **MainMenuActivity**. All of the other activities communicate with each other using an intent object as well (each dotted line represents a relation using an intent object).

CourseActivity class

This is the activity, which is started when the user clicks on a specific course from the courses list in the **MainMenuActivity**. The **CourseActivity** gets an Intent object from the previous activity, holding information to populate the sections: Course announcements, session plan, Course info and Participants. Each of them has its own section/tab, which can be seen in figure 46.

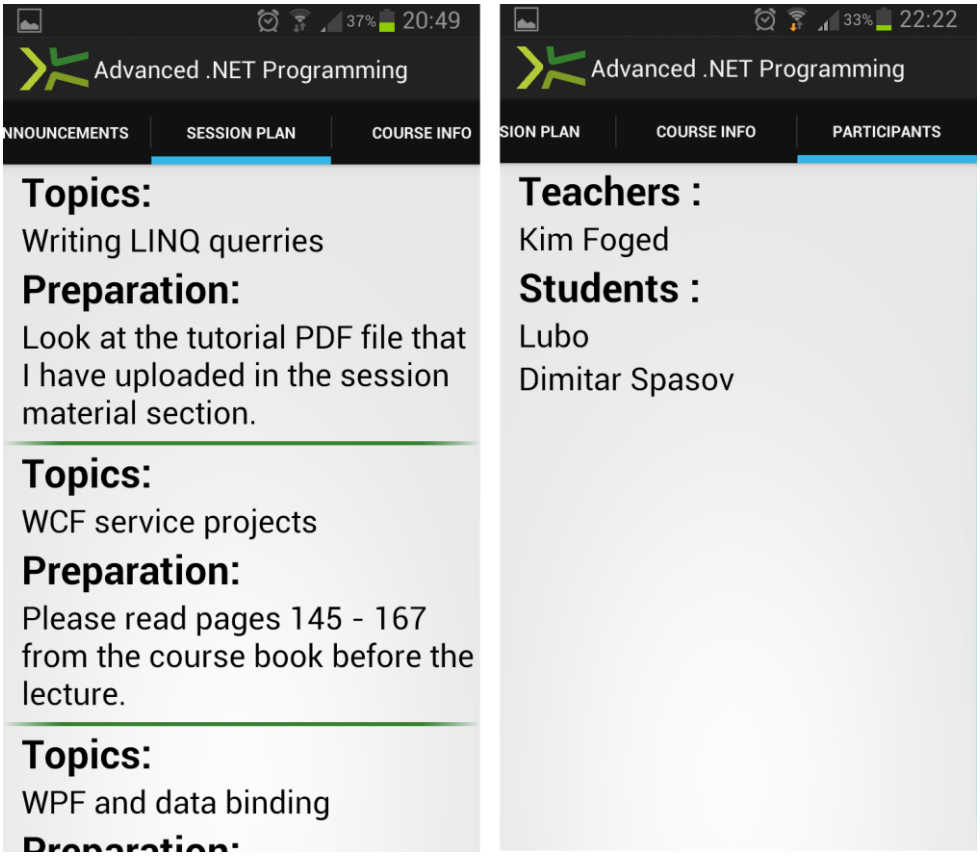


Figure 46 - Course menu

Implementation

This section of the report describes, how we went about creating the core functionality of the most important features corresponding to the most important use cases. Along with the descriptions, we are arguing on the choices we made during the construction of the system.

The backend

Creating an event

The creating of an event in the Mobile Studienet system is both simple and a bit complex at the same time. On figure 39 that we saw a bit earlier, it is shown that the event creating wizard requires for the staff member to select: the date and time of the event, select to whom it is applicable, enter the title and text and identify himself as the sender of this event. The Applies to option for example, that allows the sender to select if the event will be applicable to a whole program of students, to all students attending a specific course, or to a single student, may look simple and obvious at first, but it's not.

First things first. Let's follow the events that occur when the staff member presses the "Add" button to the Create event wizard.

The first block of code that gets executed is shown on figure 47. Its job is to get the information about the event that was entered in the wizard and manipulate it in such a way, that it will be suitable to enter in the database.

```
DateTime? date = dateTime.SelectedDate;
DateTime now = DateTime.Now;
if (date != null && eventTitleTextBox.Text != "" && eventTextTextBox.Text != "" && senderComboBox.SelectedIndex != -1 && appliesToComboBox.SelectedIndex != -1)
{
    String selectedDate = date.Value.ToString("yyyy-MM-dd") + " " + hoursComboBox.Text + ":" + minutesComboBox.Text + ":00";
    String modified = now.ToString("yyyy-MM-dd HH:mm:ss");
    stu.CreateEvent(selectedDate, eventTitleTextBox.Text, eventTextTextBox.Text, staffIndexList.ElementAt(senderComboBox.SelectedIndex).ToString(), modified);
```

Figure 47 - Code for creating an event entry in the database

In this part we have two dates. The first one is when the event deadline is, and the second is the system date and time that are of the Mobile Studienet. By having the second date we can know when was it created, or last edited. Then we have to format them in such a way that will be suitable for the database fields that hold them. The format of the date must look like this: **2013-12-13 11:59:00.** Afterwards we call the **CreateEvent** method that is from the WCF service, and in the parentheses we give the new formatted date and all the rest of the fields that were populated in the wizard.

```
public string CreateEvent(string dateAndTime, string title, string text, string staffID, string dateModified)
{
    try
    {
        using (MobileStudienetEntities entities = new MobileStudienetEntities())
        {
            EventTable table = new EventTable
            {
                dateAndTime = Convert.ToDateTime(dateAndTime),
                title = title,
                text = text,
                staffID = Convert.ToInt32(staffID),
                dateAndTimeModified = Convert.ToDateTime(dateModified)
            };
            entities.EventTables.Add(table);
            entities.SaveChanges();
            return "Success";
        }
    }
    catch
    {
        throw new Exception();
    }
}
```

Figure 48 - CreateEvent function

After we have called the **CreateEvent** function from the code in figure 47, and passed the arguments to it, we come to the code in figure 48. What the **CreateEvent** function does, is that it takes as arguments all of the data that the user entered in the wizard, except the "Applies to" option.

First is the **dateAndTime** argument. It represents the selected date and time values that the staff member selected in the wizard. Next are the **title** and **text** arguments, which represent the title and body text of the event. After that is the **staffID**, it is a unique identification number that corresponds to the selection of the "Sender" field of the wizard. And last comes the **dateModified** argument, which indicates the local time at what the event was created or last edited.

These are the only fields we need in order to insert the event in the database. This is done by creating an instance of an object that represents the Entity Framework in the system. This instance of the object, will provide us with the necessary commands that will allow us to insert the information in the database and then save the changes we have mage.

The next step in the process is to create an instance of the **EventTable** model class that is an object which represents the **EventTable** in the database. It is the object called **table**. That way we can map the arguments that we gave the function to the fields of the database table.

After we have initialized that object, we have to set its properties to be equal to the arguments we passed to the function from the beginning. This would make the **EventTable** object full of the information that the staff member entered in the wizard.

After that step is done, the last thing to do is to add the information to the database table using the **entities.EventTables.Add(table);**. Which adds the **table** object with all of its fields into the database as a new entry of information. And at last we call the **entities.SaveChanges();** command, which commits the new changes that have been made to the database.

```
public partial class EventTable
{
    public EventTable()
    {
        this.EventStudentBTs = new HashSet<EventStudentBT>();
    }

    public int id { get; set; }
    public System.DateTime dateAndTime { get; set; }
    public string title { get; set; }
    public string text { get; set; }
    public int staffID { get; set; }
    public System.DateTime dateAndTimeModified { get; set; }

    public virtual ICollection<EventStudentBT> EventStudentBTs { get; set; }
    public virtual StaffTable StaffTable { get; set; }
}
```

Figure 49 - EventTable model class

On figure 49 is the code of the EventTable model class that was generated by the Entity Framework. As mentioned before, it represents the EventTable of the database, and it is the only thing that allows us to access this table in the database. As we can see on this picture, this model class has exactly the same fields as the database table, that's how the mapping between the code and the database table is done. After we called the

entities.SaveChanges(); command, the Entity Framework basically gets the values of the fields of the model class, and inserts them into the corresponding fields into the database.

Now let's turn our attention to the "Applies to" option in the create event wizard. In this case the "Program" radio button is selected. This means that we have to find all of the students that are enlisted for this program, and associate them with the event that was just created.

```
if (selectedRadio == "Program")
{
    foreach (var item in stu.GetAllStudents())
    {
        if (item.programID == programIndexList.ElementAt(appliesToComboBox.SelectedIndex))
        {
            stu.AssignEventToStudent(eventID.ToString(), item.id.ToString());
        }
    }
}
else if (selectedRadio == "Course")
{
    foreach (var item in stu.GetAllStudentsForCourse(courseIndexList.ElementAt(appliesToComboBox.SelectedIndex).ToString()))
    {
        stu.AssignEventToStudent(eventID.ToString(), item.id.ToString());
    }
}
else
{
    stu.AssignEventToStudent(eventID.ToString(), studentIndexList.ElementAt(appliesToComboBox.SelectedIndex).ToString());
}
```

Figure 50 - Algorithms for the "Applies to" option of the Create event wizard

Figure 50 shows all of the logic, which stands behind finding all of the students for a program, course, or a single specific student.
Basically what this piece of code does, is that it has a check which finds out what the staff member creating the event selected: the Program radio button, the Course radio button or the Student radio button. Depending on that selection, then the corresponding code is executed. If the staff member selected the Program radio button, the code between the curly braces in the first part of the check will be executed. What it does is that it goes into the **StudentsTable** in the database and looks through each student and compares if the program selected in the wizard corresponds to the program the student is enlisted for. If that is the case then it gets the student and assigns the event to that student. This happens for all of the students that are enlisted for the selected program.

The same logic is applied if the Course radio button is selected. The third scenario however does not require that much work. It just assigns the event tot the student that is selected through the wizard, if the Student radio button was active.

After finding all of the students for this program, we have to call a special function that will associate them to the event. The function depicted on figure 51 has to be called once for every student that has to be associated with the event in order for the students to receive the new event notification on the client side. In general, what this function does, is that it takes only two arguments. One of the arguments is the unique identification number of the event we just created represented by the **eventID** argument, and the second one is the unique identification number that corresponds to the current student represented by the **studentID**. This function performs the same main actions as the one that we described above. The only differences is that it takes different arguments, and uses a different model class from the Entity Framework.

```
public string AssignEventToStudent(string eventID, string studentID)
{
    try
    {
        using (MobileStudienetEntities entities = new MobileStudienetEntities())
        {
            EventStudentBT table = new EventStudentBT
            {
                eventID = Convert.ToInt32(eventID),
                studentID = Convert.ToInt32(studentID)
            };
            entities.EventStudentBTs.Add(table);
            entities.SaveChanges();
            return "Success";
        }
    }
    catch
    {
        throw new Exception();
    }
}
```

Figure 51 - Assign event to student function

This time it uses an **EventStudentBT** model class, which represents the **EventStudentBT** table in the database. As described earlier in this report this table holds the association information between the students and events in the Mobile Studienet system.

The visual representation of the **EventStudentBT** table in the database can be seen on figure 52. Here we can see that the student with unique identification number **1** is associated to events with identification numbers: **56, 57, 58, 96** and **98**. This means that the student can see this events in the Mobile Studienet application installed on his smartphone.

	id	eventID	studentID
1	55	56	1
2	56	57	1
3	57	58	1
4	96	96	1
5	98	98	1
6	99	98	15

Figure 52 - EventStudentBT holding associations between students and events

At the same time we can see that event with identification number **98** is associated with student **1** and student **15** which means that both students can see the same event in their event lists on the devices. This allows us to control who sees the events and who doesn't. It is good, because this way we present a very flexible way for the staff members to associate events to the right people.

Now comes the part where we have to make this information available for the clients to see. For this to happen, we have to make the WCF web service, to provide some kind of way for them to get the information from the server. In order to do that, we have to make the function that will give the information to the clients exposed, so that it can be called over the network. On figure 53 we can see a function called

```
[OperationContract]
[WebInvoke(Method = "GET",
    ResponseFormat = WebMessageFormat.Json,
    BodyStyle = WebMessageBodyStyle.Wrapped,
    UriTemplate = "events_for_student/id={id}")]
List<EventTable> GetAllEventsForStudent(string id);
```

Figure 53 - GetAllEventsForStudent function that is exposed to the network

GetAllEventsForStudent. Before that function there are some properties set to that function. One of the properties is the OperationContract property. This property is the thing that makes this function exposed over the network, so that it can be called by the clients of the system. The second property, which is the WebInvoke property, specifies some special settings to that function.

The first one: **Method = "GET"** specifies to that system that this function will only allow the client to get information from the server. Or in other words that the client will not be allowed to write or manipulate information using this specific function.

The second setting **ResponseFormat = WebMessageFormat.Json** tells the service that the message this function is going to return to the client is going to be on the type of a JSON message.

The third setting **BodyStyle = WebMessageBodyStyle.Wrapped** sets the contents of the message to be packed as tightly as possible. That means that there will not be any useless spaces or new lines in the message, so that it will be as compact as possible.

And the fourth final setting to this function **UriTemplate = "events_for_student/id={id}"** is the signature that the client uses to call this specific function over the network. It is the thing that differentiates this function from the rest functions that are created in the service. When the client calls this function using this signature, in the **{id}** part of the signature, he passes the unique identification number for the student that is logged in the application. That way the function will know for which student to get the corresponding events. When the **{id}** part of the signature is substituted with the id of the student the signature should look like something like this: **"events_for_student/id=1",** for a student with an id equal to 1. Then the service will recognize that the signature contains the number, and it will extract it and use it as the argument to the function. The above function however, is just an abstract version of the function in figure 54 it depicts only what the function header looks like, but does not have a body. It is used like a link to the real function. The function in figure 54 however has the body and that is the code that will get executed, when the client uses the signature above.

```
public List<EventTable> GetAllEventsForStudent(string id)
{
    try
    {
        int studentId = Convert.ToInt32(id);

        using (MobileStudienetEntities entities = new MobileStudienetEntities())
        {
            return entities.EventStudentBTs.Where(s => s.studentID == studentId).Select(i => i.EventTable).ToList();
        }
    }
    catch
    {
        throw new Exception();
    }
}
```

Figure 54 - Full GetAllEventsForStudent function

When this function already has the unique id of the student that wants to see his events, it uses it to search in the database table **EventStudentBT** through the Entity Framework. It gets all of the events identification numbers that are associated to this student. Then it uses these identification numbers to find the full event information from the **EventTable** in the database. And then when it has all of the event information, it returns it to the client in the form of a list of events.

The client uses the same technique, when it comes to getting the rest of the information it needs. The only difference is the signatures that it uses to call the functions in the web service, and the model classes the WCF service uses in order to access the information from the database.

Fail proofing

Things can always go wrong with software, that's why we had to incorporate a lot of fail checks. If we did not do that, sometimes the system crashed, was unable to work properly, or was not producing the right data. The checks are represented by message boxes that pop up if the conditions are not right.

```
else if (!go)
{
    MessageBox.Show("Student number or CPR already exist!", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
}
else if (passwordTextBox.Text.Length < 6)
{
    MessageBox.Show("Password must not be less thatn 6 characters!", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
}
else
{
    MessageBox.Show("Some of the fields are empty!", "Warning", MessageBoxButton.OK, MessageBoxImage.Warning);
}
```

Figure 55 - Message boxes for different warning scenarios

One of the most common checks we did, was to check if some fields of the Admin Panel wizards are empty and notify the staff member. That way there would be no empty entries in the database, and the system will be clean of bad data. We also incorporated other types of checks. For example the student number that is given to students when they are entered into the system, must be exactly 6 numbers long i.e. **142323.** The same goes when they are given the CPR number, which must be exactly 10 numbers long.

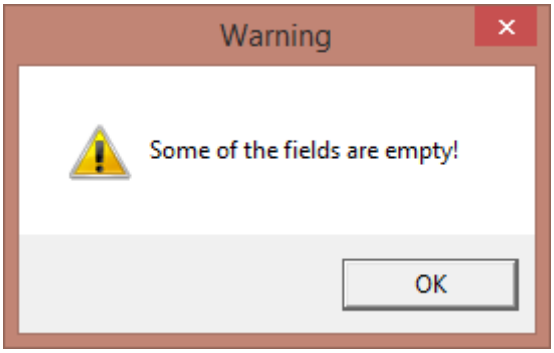


Figure 56 - Empty fields warning

Another check we have is that the student numbers must contain only numbers and no characters. We are also checking that the password to their accounts cannot be less than 6 characters, because then the security is much worse.

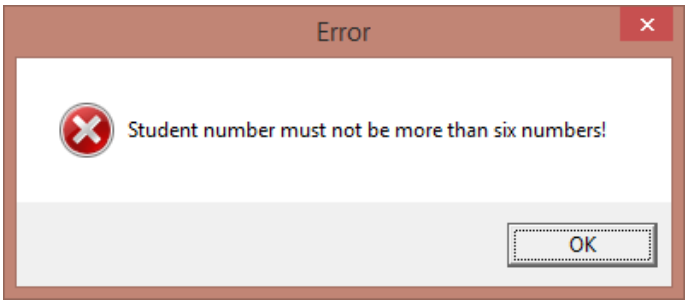


Figure 58 - Student number length error

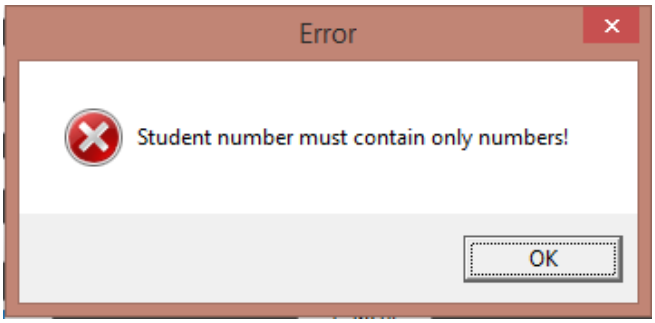


Figure 57 - Student number character error

Checks like these keep the user of the Admin Panel informed if they have done something wrong, so that they know what to change. Some of them act as warning messages, and some of them act as errors. With these messages, we can keep the system fail proof, and protected from crashing in case something out of the ordinary happens. Some of them are a result of the testing that we have performed, which showed some weaknesses, and made us fix the holes in the security of the system.

The frontend

Polling data

Getting the data from the backend and notifying the user for updates is quite straight forward, and yet not so easy to be implemented. As mentioned earlier in the design section of the report, in order for the client to update the current data on the mobile device, he has

to first get the new data from the server. The diagram below illustrates the logic behind the polling mechanism that is used in the Mobile Studienet.

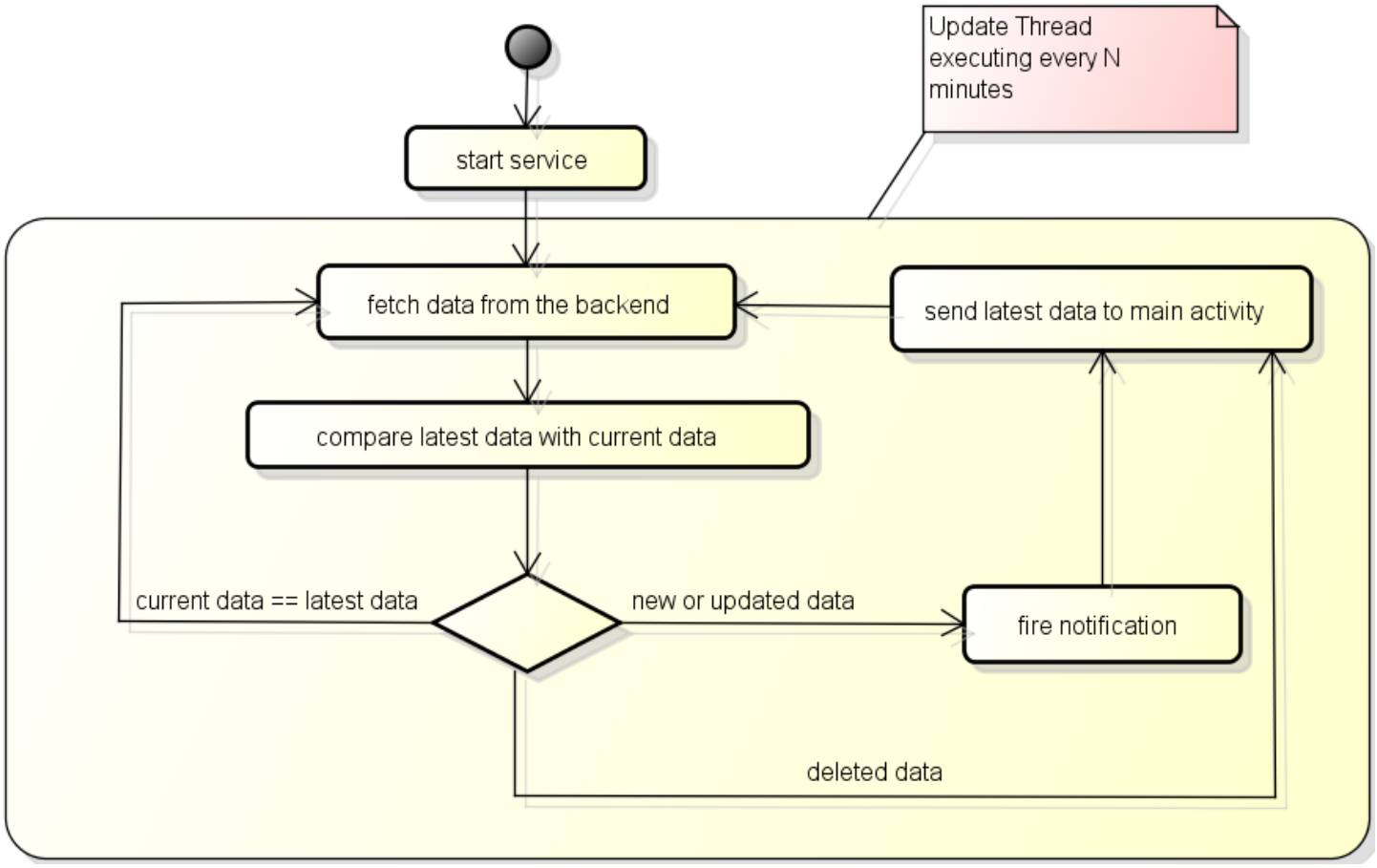

Update Thread executing every N minutes
start service
fetch data from the backend
send latest data to main activity
compare latest data with current data
current data == latest data
new or updated data
fire notification
deleted data

Figure 59 - PollingService activity diagram

All of the data retrieved from the backend, goes through the **PollingService**. That way, we avoid possible conflicts of data mismatch. First, the **PollingService** is started and when that happens, the update thread that is inside the **PollingService** is also started. All the magic happens exactly within the update thread, where the latest data from the server is fetched in a JSON format, and decomposed to be placed in the model classes. Using the model classes is necessary, so that the latest data can be converted to become compatible with the current data in the application. When that is complete the comparison between both sets of data commences. The outcome of that comparison determines what is going to happen next:

a) If there are new or updated entries. In that case a notification is fired, notifying the user, and the latest data is sent to the **MainMenuActivity**, where the UI is updated.
b) If there are no new or updated entries. In that case, after N minutes, data is fetched again and the logic repeats itself.

c) If there is deleted entries. In that case the user does not get a notification, and what happens, is simply that the UI removes the entry that has been deleted.

Now that we have a good grasp of the logic behind the polling, we can take a look at how it is implemented within the Android project.

Starting the PollingService

```
132    @Override
133    protected void onCreate(Bundle savedInstanceState) {
134        super.onCreate(savedInstanceState);
135        setContentView(R.layout.activity_main_menu);
136
137        initializeLists();
138
139        updateReciever = new UpdateReciever();
140          IntentFilter intentFilter = new IntentFilter();
141          intentFilter.addAction("UPDATE");
142        registerReceiver(updateReciever, intentFilter);
143        serviceIntent = new Intent(MainMenuActivity.this, PollingService.class);
144        startService(serviceIntent);
```

Figure 60 - MainMenuActivity onCreate() method

Starting the **PollingService** happens in the **onCreate()** method of the **MainMenuActivity**. In Android every Activity has an **onCreate()** method, which is executed when the activity is started. Initialization is a common thing to do in the **onCreate()** of an activity and considered a good practice in Android development.
Taking a deeper look in what happens within the method, we notice the **setContentView(…)** method, which basically sets the graphical layout for the activity. The argument the method takes, is an integer, which represents an address in memory pointing to the place where this layout is stored. Next is the **initializeLists(),** a method that initializes several ArralyLists which later will be used for holding the data objects, received from the PollingService. In order for the **MainMenuActivity** to be able to receive Intent objects from the **PollingService** it needs to "listen" somehow for new incoming Intent objects. One of the best practices to do so is by using Android's **BroadcastReceiver** class, because it has a more secure way of transferring the data. In our case we have an **UpdateReceiver** class that extends **BroadcastReceiver** and is used for consuming the incoming Intent objects from the PollingService. In order for the **UpdateReceiver** to know which Intents to consume it's given an **IntentFilter** as a parameter with a specific action (in our case that is UPDATE). So what does this actually mean? It means that the receiver will accept Intent objects, which have the same action as the receiver (pretty much filtering what we want to receive). After this is done, the receiver is registered to the **MainMenuActivity** by the **registerReceiver()** method that takes the actual receiver and the **IntentFilter** as parameters.
Now that the **MainMenuActivity** is setup and prepared to receive Intent objects from the **PollingService**, we can actually start the **PollingService** itself. That is done by the **serviceIntent** which specifies the start and endpoint of the action that is about to take place

(notice that it takes the **MainMenuActivity.this** and **PollingService.class** as parameters) so when the **startService()** method is called, it knows what service to start.

Fetching and comparing data from the backend

```
123     @Override
124     public int onStartCommand(Intent intent, int flags, int startId) {
125         ClearNotificationsReceiever receiver = new ClearNotificationsReceiever();
126         IntentFilter intenFilter = new IntentFilter();
127         intenFilter.addAction("clear");
128         registerReceiver(receiver, intenFilter);
129         updatedEventIds = new ArrayList<Integer>();
130         courseIds = new ArrayList<Integer>();
131         connect = new Connect();
132         connect.execute();
133 
134         updateThread = new Thread(new Runnable() {
135 
136             @Override
137             public void run() {
138                 while (true) {
139                     getLatestEvents();
140                     checkEvents();
141                         if(hasEventChanged){
142                             sendLatestEvents();
143                         }
144                     getLatestAnnouncements();
145                     checkAnnoucements();
146                         if(hasAnnouncementChanged){
147                             sendLatestAnnoncements();
148                         }
149                     try {
150                         Thread.sleep(updateTime);
151                     } catch (InterruptedException e) {
152 
153                         e.printStackTrace();
154                     }
155                 }
156             }
157 
158         });
159 
160         updateThread.start();
161         return START_STICKY;
162     }
```

Figure 61 - PollingService onStartCommand() method

When a service in Android is started, the **onStartCommand()** method is executed. It is the access point for the Intent that we just sent from the **MainMenuActivity**.
The first thing that happens here is registering a receiver similarly to the receiver in the **MainMenuActivity**, only this time we register the receiver to listen if the clear notifications button is pressed (this is the button that becomes visible and active when notifications are present on the screen). The only way the click event of the clear notifications button can be consumed is via a receiver. It is an unusual thing for Android because all of the rest types of buttons can have their own click listeners which are triggered when the user clicks them, but not this one for some reason. Next some ArrayLists are initialized, which will be used for holding id values of some particular entry objects (necessary for the comparison logic).

Assuming this is the first time the service is started **connect.execute()** is called, so it can make the very first initial fetch of data from the server, and send it to the **MainMenuActivity.** All of that happens in the **connect.execute(),** and it's necessary for the logic to work because without that initial fetch, we won't have any current data to compare to the latest data. The next step is to initialize our update thread that will be doing all the fetching every N minutes. The **run()** method of the thread is what is executed when the thread is actually started. So let's take a closer look behind the way the thread is constructed. First we call the **getLatestEvents()** method. In it, we fetch the latest events from the backend server in a JSON format and unpack it in the **Event** model class. Once the latest event data is stored in the client application using the **Event** model class, we can compare it with the current event data. That happens in the **checkEvents()** method. Three are the possible outcomes of this method:

a) It finds new or updated events, fires the notification and sets the **hasEventChanged** flag to true. Because the **hasEventChanged** flag is set to true, the latest events are sent to the **MainMenuActivity** with the **sendLatestEvents()** method.

b) No new or updated events found nothing happens.

c) No new notifications. The UI removes the entry that has been deleted.

In a similar fashion, we get the latest announcements from the server with the **getLatestAnnouncements()** method and compare with the **checkAnnouncements()** method. The outcome is identical to the outcome of the fetching of the latest events.
Moving further down we set the time frame in which we want our **UpdateThread** to execute (those N minutes we talked about so much already). Now that we are done with the initialization of the thread, we can start it by calling **updateThread.start().**
The **START_STICKY** that the **onStartCommand()** returns, is basically a flag telling the Android OS that we want our service to run forever (even if for some reason like low memory the operating system stops the service the Android OS will restart the service).

Testing and results

General testing

In this project the testing period began almost instantly, when we started implementing the functionality we have planned from the beginning. This is because in order to verify that something that has just been created works, we have to test almost every time there is even minor changes to the code.

For example, when we created the tables in the database and all of the constraints to the fields in each table, we immediately started inserting some information in order to test out if those constraints work properly. When we created the WCF web service, we did not continue our work before we made sure that the connection functions properly between the device and the server.

White box testing

Along with the general testing we did during the implementation phase of the project, we also incorporated white box and black box tests on the system. The white box testing is a special type of testing, where the people that are performing the tests on the system, have deep knowledge of it and its internal workings. This is usually done by the people who have designed the system themselves.

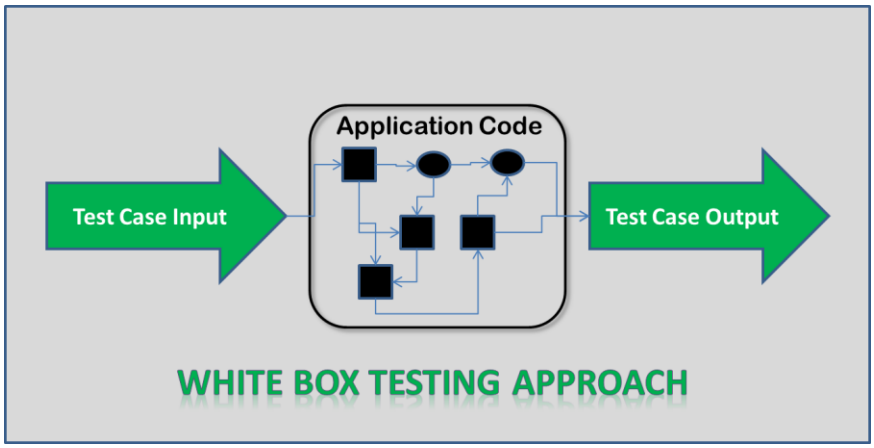


Figure 62 - White box testing

In our case, almost at the end of our implementation we started focusing on the final testing of the product. The first type of tests we did was of the white box type. We started carefully going through our code and one by one we tested every function of our code to see how it processes the input and what output it gives back. This way we analyzed if the function has potential weak points, and fixed them if we found any. Also by doing this, when going through the code, sometimes we even found better ways to improve our algorithms.

This type of testing not only allowed us to fix errors, but even to refine our strategies of the system functionality. Of course, after changing something, we also had to test the rest of the functionality again, because sometimes when fixing an error, it can make a working part of the code faulty.

For testing and debugging the code, we used the same programming tools that we used to develop the system. They have built in debuggers and error reporting features, which allowed to fix our code with ease. For the backend, we used the Visual Studio to tell us what types of errors we had and to suggest how we could fix them.

The error report on figure 63 shows that there was a problem with deleting an event from the database. This could have been only because the event was assigned to one or a number of students. This is because we cannot delete an entry from the database, which is being referenced to by other entries.

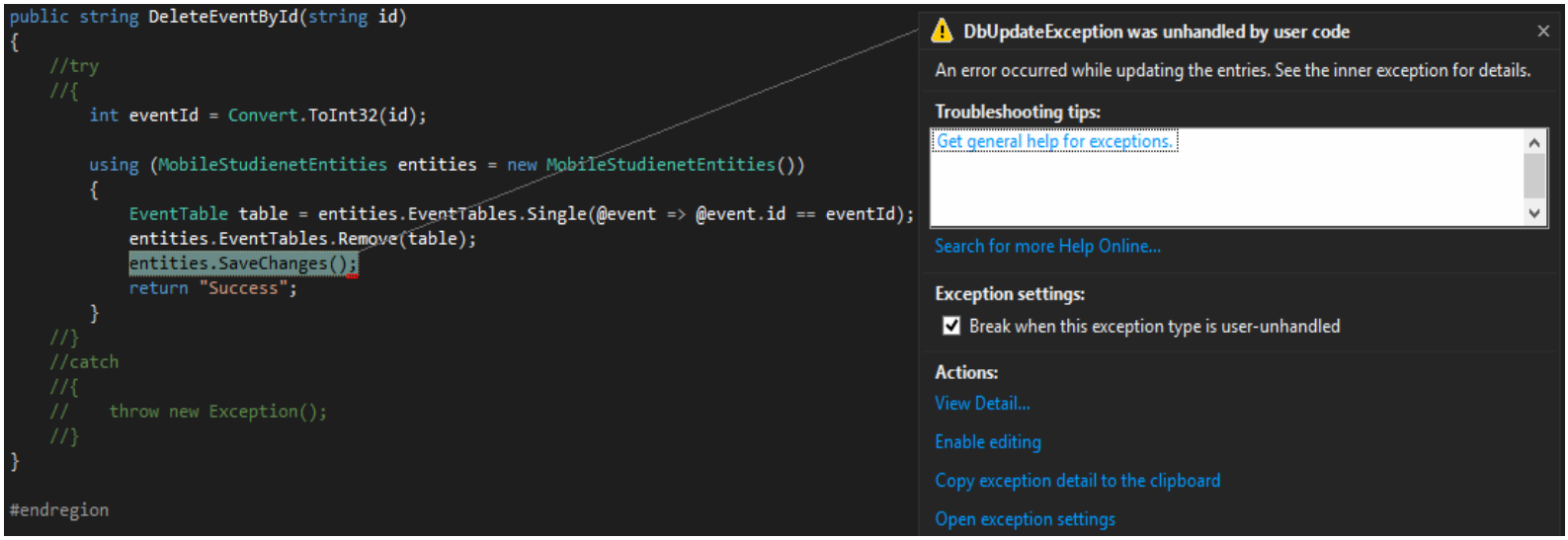


Figure 63 - Error report from a program crash

For the frontend, we used a tool called LogCat. It is a tool that comes with the Android SDK, and basically represents a console that prints out messages, which contain information on how the mobile application works when it executes certain parts of the code. Along with the system messages of the device, we can add our own messages to be printed, in case we want to test something specific, or just show a value of a variable. A snippet of the LogCat is shown on figure 64.

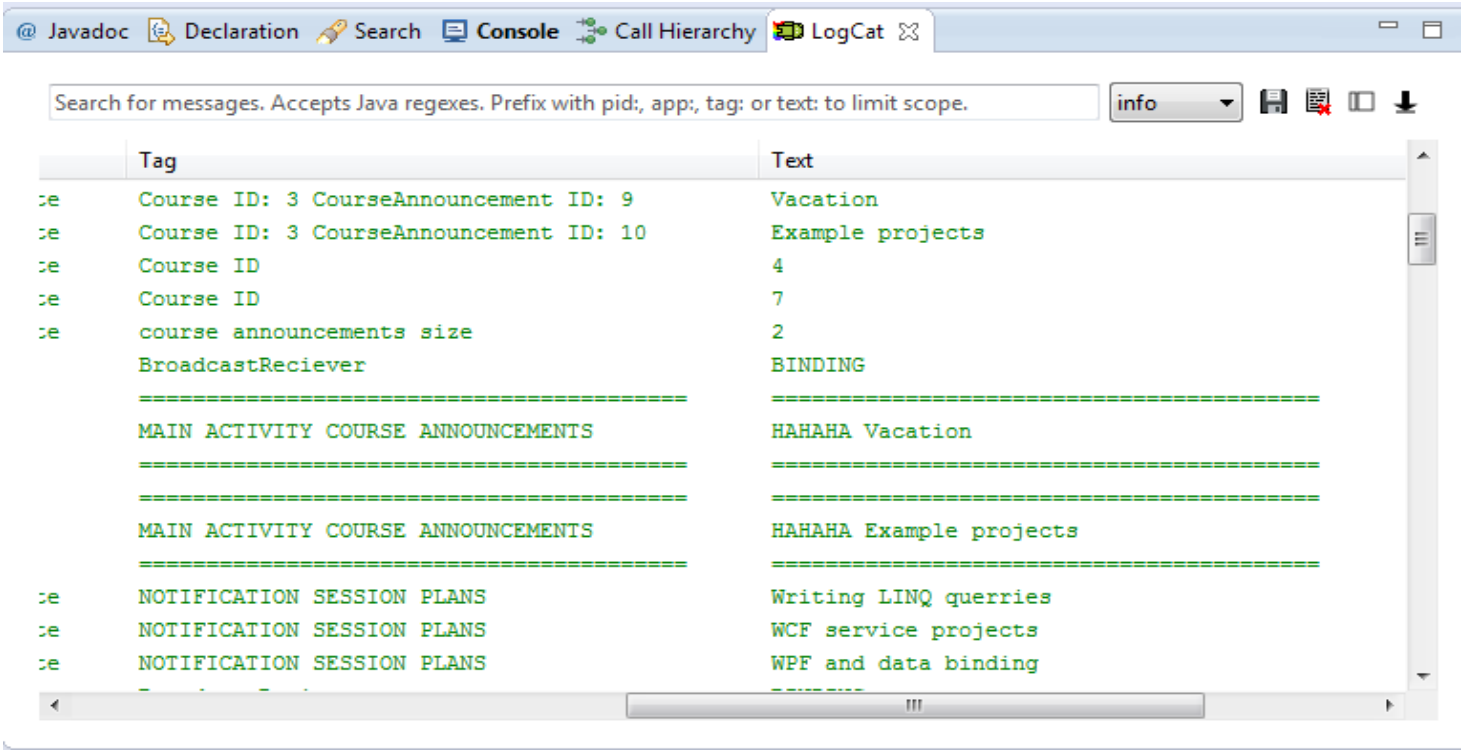


Figure 64 - Android SDK LogCat tool

Black box testing

On the contrary of the white box testing, the black box testing, is a type of testing where the person that performs the tests to the system, has no knowledge of how the system works at all. It is usually done by third party people invited just to test the system out, or by the clients of the system. This allows for the tester to use the system in such a way, like he would use it in his everyday life. This will test the system in the way it is meant to be used. Mostly in this type of testing there shouldn't be much errors to report, but no one ever knows where the bugs are hiding.

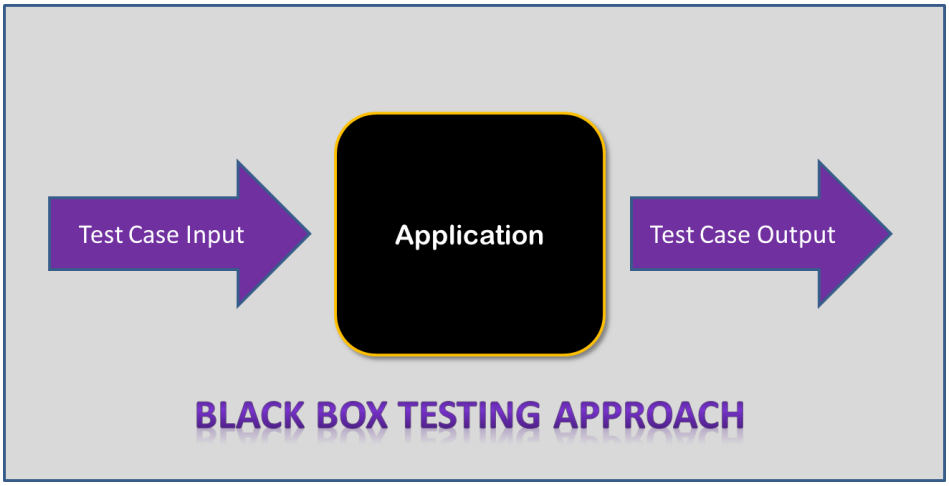


Figure 65 - Black box testing

That's why in our case for the Mobile Studienet, we invited some friends over, which are also students at VIA to test the application as normal users. We installed the application on bunch of android devices with different hardware specifications (CPUs, display resolutions, RAM memory, etc.), and gave them a hands-on with the system. They used the Mobile Studienet system like a normal student would, and tried out every possible functionality we have incorporated in it. With a thorough use of the system, together we all managed to find a couple of more errors and text typos, but overall the system was functioning fine and with no hiccups.

Discussion

During the project period, we have successfully managed to implement all of the main requirements for the Mobile Studienet system, but since we had limited timeframe, the system still has a very big potential of improving.

Possible improvements in the android application:

- We can further improve the information processing algorithms that receive and decode the data from the server.

- The user interface can be improved even further for even better looks and user friendliness. For example, we can implement user settings, where the user will be able to choose the sound tone for the notifications, or for which things to receive notifications etc.

- In the future we could make so that the student will be able to change his or hers password through the mobile application.

Possible improvements in the server side:

- We could make a more sophisticated admin panel, which would be able to show the information in a more structured way, and maybe incorporate log-in function as well.

- It will be possible to improve the management of information, and to make it easier to insert or to edit.

Improvements in general to the system:

- Future Stads access, so that the students can track their grades.

- The Mobile Studienet can be developed for other operating systems like iOS for Apple smart phones, or Windows Mobile for Windows powered devices.

- The system can be made to support different languages, so that the users can select their native languages if they prefer.

- We can improve the application so to support the session materials section of the Studienet, so that the users will also be able to view the materials their teachers upload.

Conclusion

In conclusion to this project, we can proudly say that the requirements for the Mobile Studienet system has been successfully met, and the system is fully functional at its current state.
We are very satisfied with the results of the system, and we believe that if it was to be implemented in reality, it would serve a very good purpose to all students of VIA University College.

During the project period, the team managed to learn some new skills and programing technologies. At the beginning we did not have all the necessary knowledge, but we knew that this was the perfect opportunity to gain some more experience and improve ourselves as developers, and at the end we came out to be right. We learned a lot in the sphere of Android application development and Microsoft based technologies.

Overall, working towards creating the Mobile Studienet system has been a very fun and fruitful experience. We believe that the idea we presented with this project is really good, and if you look one last time at the survey results from the students, you will see that it is not only we who think so.

Appendices

Appendix 1 – Document references

[1] FreeOnlineSurveys is a free web solution that allows any user to freely create their desired survey, and then distribute it anyway they could possibly want using the internet. Official webpage: http://freeonlinesurveys.com/

[2] Reverse engineering is the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation. It often involves disassembling something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) and analyzing its components and workings in detail - for either purposes of maintenance or to support creation of a new device or program that

does the same thing, without using or simply duplicating (without understanding) the original. Original article: http://en.wikipedia.org/wiki/Reverse_engineering

[3] The Google Play Store is an application that is installed by default into every Android powered smartphone. It is the marketplace from which a user can download all kinds of applications, music and books to his or her device.

[4] ASP.NET is a technology included in the Microsoft .NET software package. This package is a framework that is needed to create and run programs that are written in the C# language. It also contains a lot of other technologies like the WCF project we are using, and the Entity Framework that is connecting our database and the WCF service. Original article: http://en.wikipedia.org/wiki/.NET_Framework

[5] SQL Injection is a way for someone with malicious intentions that is using the application to try and write SQL code in input fields that can harm the database or the entire system. Original article: http://en.wikipedia.org/wiki/SQL_injection

[6] White box testing original article: http://en.wikipedia.org/wiki/White-box_testing

[7] Black box testing original article: http://en.wikipedia.org/wiki/Black-box_testing

Appendix 2 – Detailed use case diagram description

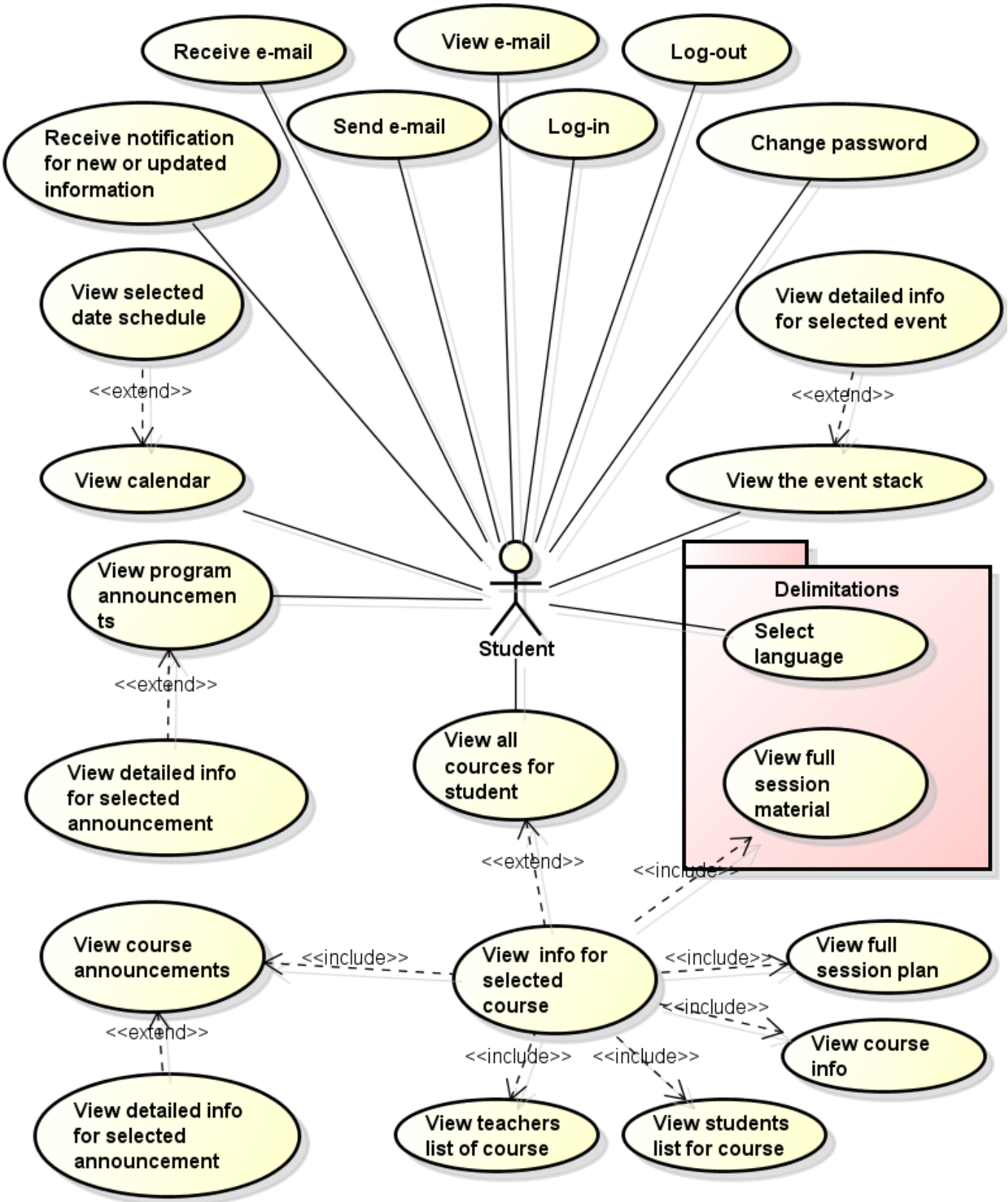


Figure 66 - Use case diagram

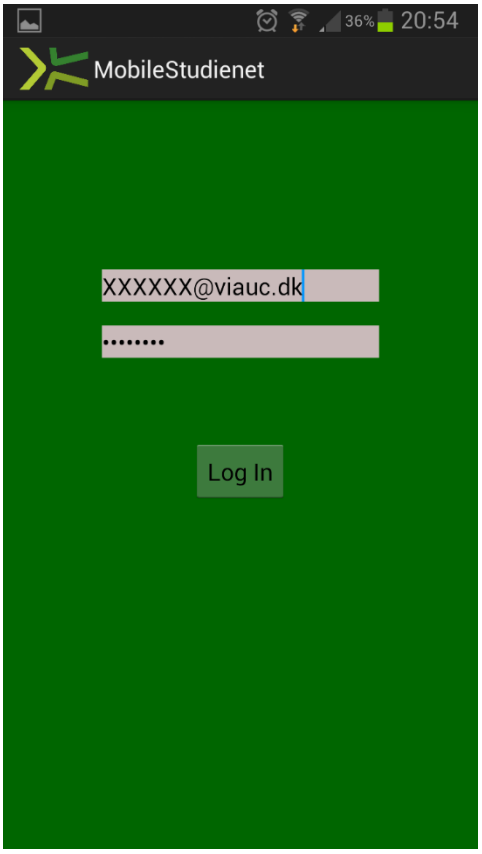
1. **Log-in:**
   Pre-conditions: The user must have an active account in the Studienet database.
   Post conditions: User has successfully logged in the Mobile Studienet.
2. **Log-out:**
   Pre-conditions: The user must have logged-in first to be able to log-out.
   Post conditions: The user successfully managed to log-out.
3. **Change password:**
   Pre-conditions: The user must remember his old password in order to change it. And he must enter the new password twice, in order to confirm it.
   Post conditions: The user has successfully changed his password.
4. **View the event stack:**
   Pre-conditions: The user has to navigate to the event stack screen.
   Post conditions: The user is able to see a list with all upcoming events.
5. **View detailed info. for selected event:**
   Pre-conditions: The user must be on the event stack screen and he must select an event from the list.
   Post conditions: User is able to see detailed information for the selected event.
6. **View courses for student:**
   Pre-conditions: Student must navigate to the courses screen.
   Post conditions: The student can see a list of all the courses he is participating in.
7. **View info. for selected course:**
   Pre-conditions: The user must have selected a course from the course screen.
   Post conditions: The user is presented with information about: course announcements list, session plan list, course info, and students and teachers list.
8. **View detailed info about a course announcement:**
   Pre-conditions: The student must navigate to the course announcements list and select an announcement.
   Post conditions: the user is presented with detailed information about the selected announcement.
9. **View program announcements:**
   Pre-conditions: The user must navigate to the program announcements screen.
   Post conditions: The user is presented with a list of all of the program announcements.
10. **View detailed info for selected program announcement:**
    Pre-conditions: The user must have selected an announcement from the program announcements list.
    Post conditions: The user is presented with a detailed information about the selected announcement.
11. **View calendar:**
    Pre-conditions: The user must navigate to the calendar screen.
    Post conditions: The user is presented with a monthly view of the calendar.
12. **View selected date schedule:**
    Pre-conditions: The user must have navigated to the calendar screen and he must have selected a certain date.
    Post conditions: The user is presented with the daily schedule for the selected date.
13. **Receive a notification for new or updated information:**

Pre-conditions: The user must have started the Mobile Studienet application at least once before to be eligible for receiving of notifications.
Post conditions: The user will start receiving notifications whenever something is added or changed in the information.

14. **Receive, Send and view email:**

Pre-condition: The user must have set up his VIA mail account into the Android built in email client, following the tutorial on the email screen of the Mobile Studienet application.
Post conditions: The user will be able to receive, send and view emails using his VIA mail account.

Appendix 3 – User manual for the Mobile Studienet system



To log-in the application the user must enter his VIA account credentials in the fields of the log-in form, and click the log-in button.

Figure 67 - Log in screen

Figure 68 - Main Menu screen

Once the user has logged in, he is redirected to the main menu of the application. From there he will be able to navigate through the different sections of the main menu (Event stack, Course, Program announcements, Calendar and Email).

If the user clicks on an event, program announcement or a course announcement, a new screen will appear, with detailed information about the selected entry. From there he can press the back button to return to the main menu/course menu.

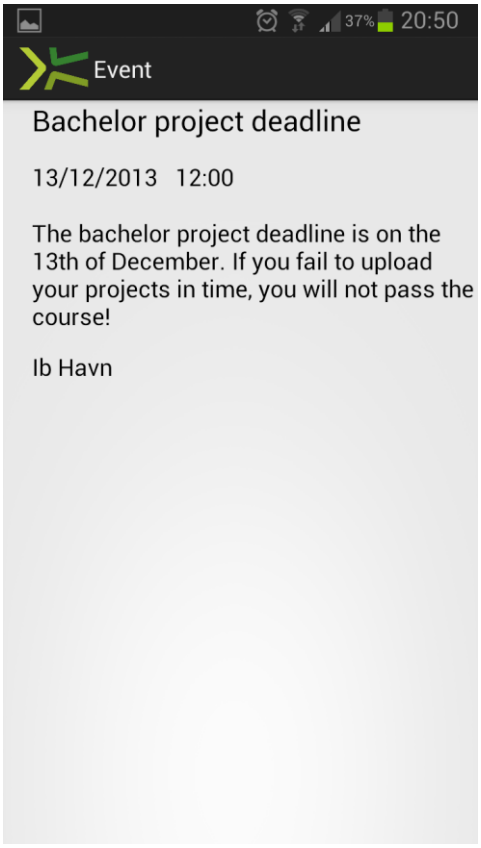

37% 20:50
Event
Bachelor project deadline
13/12/2013 12:00
The bachelor project deadline is on the 13th of December. If you fail to upload your projects in time, you will not pass the course!
Ib Havn

Figure 69 - Detailed event screen

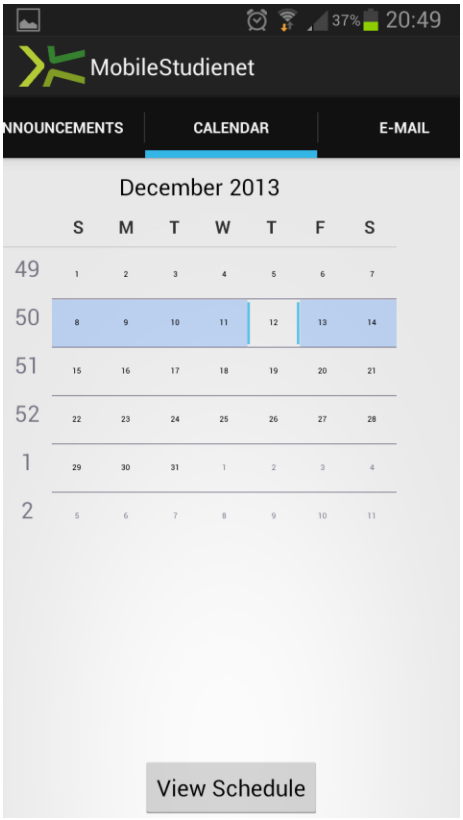


Figure 70 - Calendar screen

If the user is located on the calendar tab, he will get a visual representation of a calendar with selectable dates. If a user selects a date and presses the View Schedule button, he will be presented with a new screen, showing all of the lectures he has for this date.

Figure 71 - Daily schedule screen

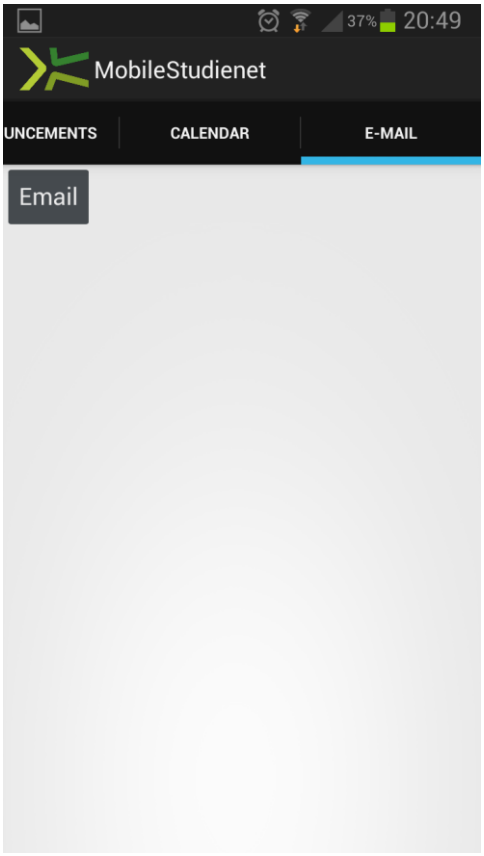


Figure 72 - Email redirecting screen

If the user is located on the email tab, he will see a button that can redirect him to the default email Android client, where he will be able to setup/view his VIA mail account.

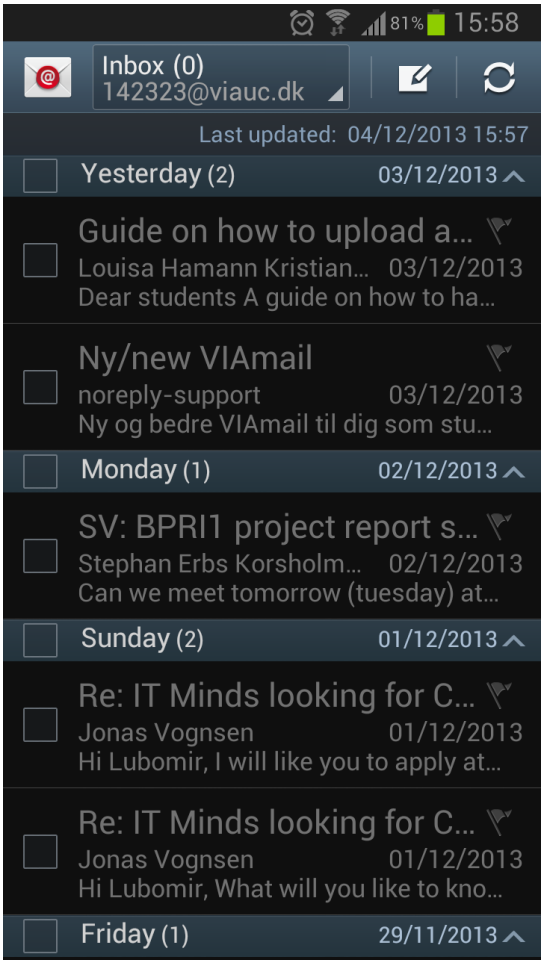

81% 15:58
Inbox (0)
142323@viauc.dk
Last updated: 04/12/2013 15:57
Yesterday (2) 03/12/2013
Guide on how to upload a...
Louisa Hamann Kristian... 03/12/2013
Dear students A guide on how to ha...
Ny/new VIAmail
noreply-support 03/12/2013
Ny og bedre VIAmail til dig som stu...
Monday (1) 02/12/2013
SV: BPRI1 project report s...
Stephan Erbs Korsholm... 02/12/2013
Can we meet tomorrow (tuesday) at...
Sunday (2) 01/12/2013
Re: IT Minds looking for C...
Jonas Vognsen 01/12/2013
Hi Lubomir, I will like you to apply at...
Re: IT Minds looking for C...
Jonas Vognsen 01/12/2013
Hi Lubomir, What will you like to kno...
Friday (1) 29/11/2013

Figure 73 - Default android email application

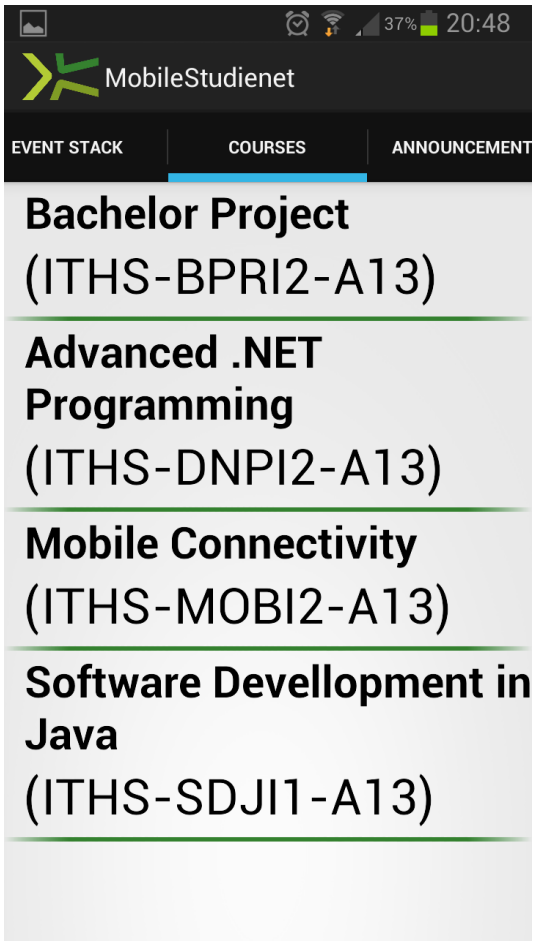


Figure 74 - Main menu courses tab

If the user clicks on one of the courses listed under the courses tab, he will be redirected to a new screen consisting of: Course announcements, Session plan, Course info and Participants list.

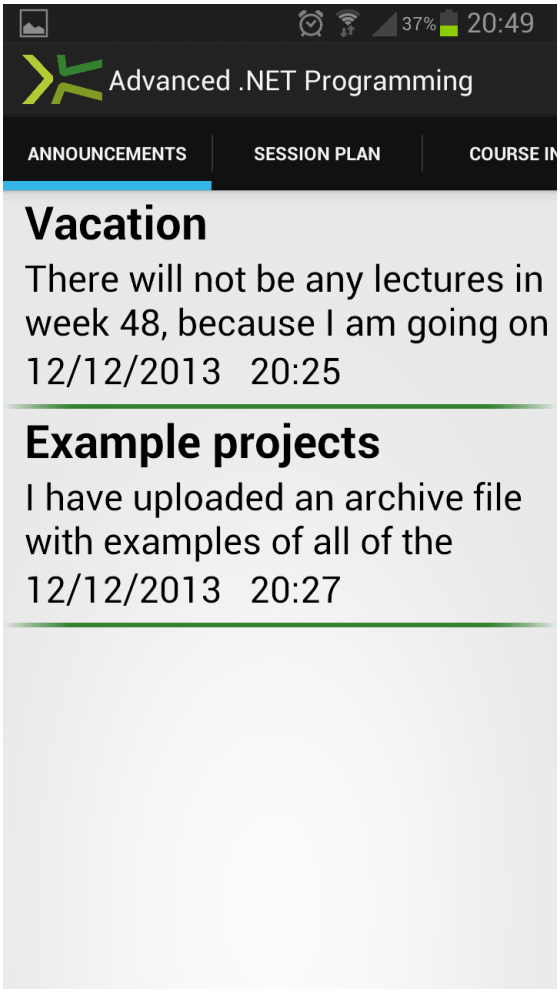


Figure 75 - Courses menu